Feng Bao   Pierangela Samarati   Jianying Zhou   (Eds.)

Applied Cryptography and Network Security

10th International Conference, ACNS 2012
Singapore, June 26-29, 2012
Proceedings (Industrial Track)

Preface

These proceedings contain the papers selected for presentation at the industrial track of the 10th International Conference on Applied Cryptography and Network Security (ACNS 2012), held during June 26-29, 2012 in Singapore. The conference was organized by iTwin, sponsored by AdNovum, and supported by Infocomm Development Authority of Singapore (IDA).

In response to the call for papers, 192 papers from 38 countries were submitted to the conference. These papers were evaluated on the basis of their significance, novelty, technical quality, and practical impact. Reviewing was "double-blind": the identities of reviewers were not revealed to the authors of the papers and author identities were not revealed to the reviewers. The program committee meeting was held electronically, yielding intensive discussion over a period of two weeks. Of the papers submitted, 33 were selected for presentation at the research track of ACNS 2012 and inclusion in Springer's LNCS 7341, giving an acceptance rate lower than 18%. In addition, the conference had 9 papers presented in the industrial track and collated in the non-archival proceedings.

The conference was also featured with 3 keynote speeches, by Moti Yung (co-founder of ACNS) entitled "Applied Cryptography and Network Security - 10 years in the past and 10 years in the future", by Peng Ning entitled "Cloud Computing Infrastructure Security", and by Hongjun Wu entitled "JH in the NIST Hash Function Competition", respectively.

There is a long list of people who volunteered their time and energy to put together the conference and who deserve special thanks. Thanks to the program committee members and the external reviewers, for all their hard work in the paper evaluation. Owing to the large number of submissions, the program committee members were really required hard work in a short time frame, and we are very thankful to them for the commitment they showed with their active participation in the electronic discussion.

We are also very grateful to all those people whose work ensured a smooth organization process: Xinyi Huang and Giovanni Livraga, Publicity Chairs, for their work in ensuring the wide distribution of the call for papers and participation; Shen-Tat Goh, Organizing Chair, as well as Lux Anantharaman and Kal Takru for taking care of the local organization; and Ying Qiu for managing the conference web site and the EasyChair system.

Last but certainly not least our thanks go to all the authors who submitted papers and all the attendees. We hope you find the program is stimulating and a source of inspiration for your future research and practical development.

April 2012 *Feng Bao, Pierangela Samarati, Jianying Zhou*

ACNS 2012

10th International Conference on Applied Cryptography and Network Security

Singapore
June 26-29, 2012

Organized by iTwin, Singapore

Sponsored by AdNovum, Singapore

Supported by

Infocomm Development Authority of Singapore (IDA)

Organizing Chair

Shen-Tat Goh	Institute for Infocomm Research, Singapore

Publicity Chairs

Xinyi Huang	Fujian Normal University, China
Giovanni Livraga	Università degli Studi di Milano, Italy

Steering Committee

Yongfei Han	ONETS, China
Moti Yung	Google, USA
Jianying Zhou	Institute for Infocomm Research, Singapore

External Reviewers

Abu Rajab, Moheeb	Agudo, Isaac	Albanese, Massimiliano
Aliasgari, Mehrdad	Asano, Tomoyuki	Athanasopoulos, Elias
Au, Man Ho	Baek, Joonsang	Bandhakavi, Sruthi
Brzuska, Christina	Cao, Jin	Chen, Jiun-Ming
Cheng, Yueqiang	Cheong, Kai Yuen	Chu, Cheng-Kang
Coisel, Iwen	Cremers, Cas	Cui, Hui
D'Arco, Paolo	De Caro, Angelo	Ding, Ning
Dong, Xinshu	Drogkaris, Prokopios	Duan, Pu
Elkhiyaoui, Kaoutar	Fernandez, Gerardo	Fett, Daniel
Fiore, Dario	Fuchs, Andreas	Fuchsbauer, Georg
Galdi, Clemente	Gorantla, Choudary	Gu, Haihua
Guo, Fuchun	Hakuta, Keisuke	Han, Jinguang
Hanzlik, Lucjan	Hao, Hao	Henricksen, Matt
Huang, Qiong	Huang, Tao	Iovino, Vincenzo
Iskander, Marian	Jee, Kangkook	Jeske, Tobias
Jin, Xing	Kalabis, Lukas	Kalloniatis, Christos
Kemerlis, Vasileios P.	Kim, Hongtae	Kim, Myungsun
Kim, Sungwook	Kohlweiss, Markulf	Kolias, Constantinos
Kontaxis, Georgios	Krzywiecki, Lukasz	Kubiak, Przemyslaw
Kuo, Po-Chun	Künnemann, Robert	Lai, Junzuo
Laszka, Aron	Le, Meixing	Lee, Hyung Tae
Lee, Younho	Leontiadis, Iraklis	Li, Fagen
Li, Juanru	Li, Xiaolei	Li, Yen-Huan
Liang, Kaitai	Lin, Jingqiang	Liu, Joseph
Liu, Junrong	Liu, Ya	Liu, Zhen
Liu, Zhiqiang	Lu, Jiqiang	Luo, Tongbo
Luo, Weiliang	Majcher, Krzysztof	Morozov, Kirill
Najera, Pablo	Neven, Gregory	Nuñez, David
Ohtake, Go	Omote, Kazumasa	Oren, Yossef
Pala, Massimiliano	Pappas, Vasilis	Patil, Kailas
Pek, Gabor	Pelosi, Gerardo	Pereira, Geovandro
Petit, Christophe	Pointcheval, David	Polychronakis, Michalis
Portokalidis, Georgios	Quaglia, Elizabeth A.	Rangasamy, Jothi
Ratazzi, Paul	Raykova, Mariana	Rekleitis, Evangelos
Rizomiliotis, Panagiotis	Roman, Rodrigo	Roudier, Yves
Roy, Arnab	Ràfols, Carla	S. Shiva, Ashwathi
Sakiyama, Kazuo	Schröder, Dominique	Schuldt, Jacob
Schwabe, Peter	Schäge, Sven	Seo, Jae Hong
Shafiq, Basit	Shin, Seungwon	Simo, Hervais
Standaert, F.-X.	Steinebach, Martin	Stopczynski, Martin
Sun, Wenhai	Sun, Xiaoyan	Ta, Vinh Thonh
Tan, Xiao	Tang, Qiang	Tillich, Stefan
Triandopoulos, Nikos	Tuengerthal, Max	Tunstall, Michael
Tzouramanis, Theodoros	Verde, Nino Vincenzo	Villani, Antonio
Visconti, Ivan	Vogt, Andreas	Wang, Boyang
Wang, Guilin	Wang, Jun	Wang, Lusha
Wang, Yifei	Wikström, Douglas	Wolny, Kamil
Wu, Wei	Wu, Wenling	Xiong, Xi
Xu, Jia	Xu, Zhaoyan	Yang, Chao
Yang, Guomin	Yap, Wun-She	Yian, Chee Hoo
Ying, Jason	Yu, Ching-Hua	Yu, Yong
Yun, Aaram	Zhan, Zhenxin	Zhang, Bin
Zhang, Jialong	Zhang, Lei	Zhang, Mingwu
Zhang, Shengzhi	Zhang, Xiao	Zhang, Yinghui
Zhao, Mingyi	Zhao, Xingwen	Zheng, Qingji
Zhong, Chen	Zhu, Youwen	Önen, Melek

Table of Contents

Security Analysis of an Open Car Immobilizer Protocol Stack

Stefan Tillich and Marcin Wójcik

University of Bristol, Computer Science Department, Merchant Venturers Building, Woodland Road, BS8 1UB, Bristol, UK
{tillich,wojcik}@cs.bris.ac.uk

Abstract. Openness is a key criterion of security algorithms and protocols which enable them to be subjected to scrutiny by independent security experts. The alternative "methodology" of secret proprietary algorithms and protocols has often ended in practical breaks, e.g. of the MIFARE Oyster cards for public transport or the KeeLoq remote control systems. Open evaluation is common for general applications of security, e.g. the NIST competitions for selection of the Advanced Encryption Standard (AES) and the Secure Hash Algorithm 3 (SHA-3). Nowadays an increasing number of embedded security applications apply the principle of open evaluation as well. A recent example is the specification of an open security protocol stack for car immobilizer applications by Atmel, which has been presented at ESCAR 2010. This stack is primarily intended to be used in conjunction with automotive transponder chips of this manufacturer, but could in principle be deployed on any suitable type of transponder chip. In this paper we analyze the security of this protocol stack. We were able to uncover a number of potential security vulnerabilities, for which we suggest fixes.
Keywords: Security, car immobilizer, algorithms, protocols, openness, analysis.

1 Introduction

Securing systems through secrecy of the involved algorithms and protocols is not always successful. Often, once the details of the algorithm have been disclosed through various channels, practical attacks quickly become possible, e.g. on the MIFARE Oyster card for the London transport system [6] or the KeeLoq algorithm used in remote control systems [7]. In contrast, subjection of cryptographic methods to public scrutiny is a widely accepted method of preventing such breaks during deployment. Prominent examples of this strategy are the Advanced Encryption Standard (AES) competition [11] and the Secure Hash Algorithm-3 (SHA-3) competition [12]. In this paper we analyze the security of a car immobilizer protocol stack which is facilitated by its openness.

A car immobilizer is a system that requires the presence of a security token (often in the form of a key fob) to allow a car to run. If this token is not present, the car's Engine Control Unit (ECU) interrupts key components like the ignition, the starter motor circuit, or the fuel pump. The communication between car and key fob is typically done via RFID, where the car is fitted with an RFID reader and the key fob contains an RFID tag. While earlier models used a static code in the key fob, modern immobilizers utilize either rolling codes or cryptography to prevent duplication of the key fob. Communication between car and key fob involves the use of a protocol stack which defines frame sizes, data formats, error detection, data transformations, etc.

An open security protocol stack for car immobilizer applications has been presented in [8]. It is mainly intended for use with specific automotive transponder chips. According to [8], the stack consists of a physical layer, a logical layer, a protocol layer, and the AES crypto layer. The physical layer deals with modulation types, data encoding, and bit timing. The logical layer defines the functional behavior of the reader and the transponder and includes communication link controls, controls configuration, setup of functional dependencies and error resolution. The protocol layer allocates data frames and buffers for reading and writing. It implements the user command interface, authentication, and key learning (*i.e.* changing cryptographic keys before and after deployment). The AES crypto layer controls the data authentication results[1]. Both physical and AES crypto layer are already industry standards. The logical and protocol layer, which are usually proprietary, are made open. This means the specification of these layers is available for inspection and modification.

The protocol stack implements a number of commands to be issued by the reader to the key fob. In most cases, the car featuring the immobilizer functionality acts as reader but the reader can also be a programming device used by the car manufacturer or distributor. The communication between reader and key fob uses the LF band at 125 kHz. In this band, the normal read range is usually very limited (commonly a few centimetres), but there are readers available which can extend it to up to one metre [3, 5] and thus allowing for attacks in close proximity of the key fob.

The command set out in the protocol stack's specification [1] encompasses eleven commands. They include reading of the key fob's unique ID (UID) and error status, initiation of authentication, setting of the used secret keys, initiation and leaving of the so-called enhanced mode (for RF communication powered by the battery), a request to repeat the last response, reading and writing of user memory as well as setting memory access protection to certain memory sections. Authentication can be configured to be unilateral (only key fob authenticates itself to the reader) or bilateral (both key fob and reader authenticate themselves to each other). If bilateral authentication is configured, some commands like reading and writing user memory can only be executed when there has been a previous successful authentication.

[1] The description of this protocol layer in [8] probably refers to the use of the AES block cipher in the execution of various commands by reader and key fob. As such it is debatable whether it constitutes a separate layer or should be considered as part of the protocol layer.

Authentication follows the challenge-response pattern [10]. The party who wants to authenticate sends out a challenge (usually a random number) and the other participant transforms the challenge cryptographically using a secret or private key and returns the response. The first party then checks this result using its knowledge of the same secret key or the according public key. The point of the challenge is to prevent replay attacks, where messages recorded from a genuine protocol run are replayed by an attacker at a later time to achieve authentication. Therefore, the challenge must be non-repeating or only repeat with negligible probability.

The investigated protocol stack has the caveat that the key fob is not expected to be able to generate challenges. This is no problem for unilateral authentication, where the challenge is generated by the reader alone, but poses difficulties for bilateral authentication. Bilateral authentication works by reusing the challenge from the reader for the challenge of the key fob. The cryptographic transformation involved in the authentication is AES encryption with one of two shared keys. Figure 1 shows the essential steps of bilateral authentication as given in [1]. Note that the complete authentication also includes the car reading the UID from the key fob in order to allow early termination of the protocol when a wrong key fob is accidentally in read range. We have omitted this part of the authentication protocol as it is not of interest from a security point of view.

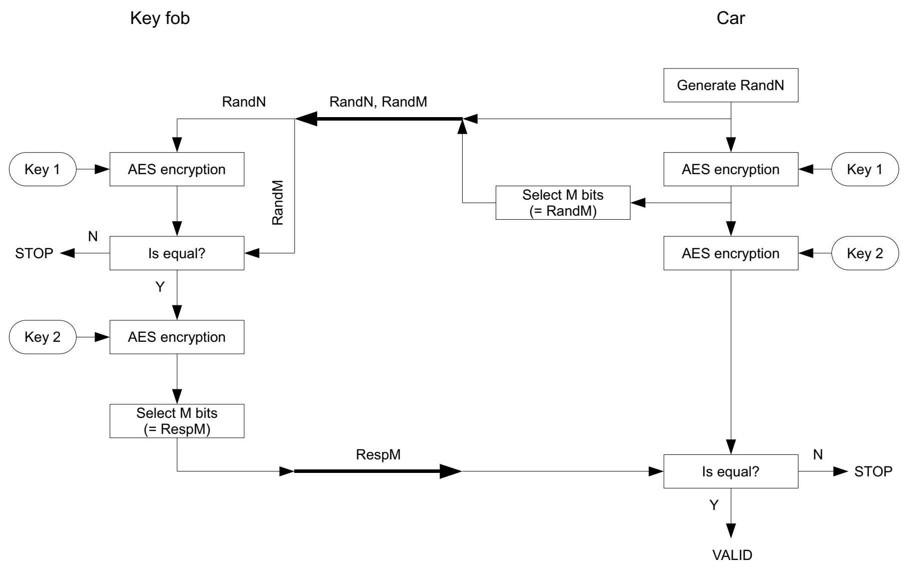


Fig. 1: Bilateral authentication between key fob and car.

Car and key fob share two AES keys (Key 1 and Key 2). The car generates the N-bit challenge RandN, encrypts it with Key 1 and selects M bits of the resulting ciphertext as RandM. N and M can be configured to be less than the AES block size of 128 bits in order to reduce communication overhead. RandN and RandM are sent to the key fob, which validates that RandM originated from RandN via encryption with Key 1. If this is successful, the car is authenticated to the key fob. The key fob uses the output of the first AES encryption as input for a second AES encryption with Key 2. As this value is not fully known to an eavesdropper (M being usually smaller than 128), it is also denoted as hidden challenge. M bits of the second encryption result are selected as RespM, which is sent to the car. The car then verifies that RespM resulted from encryption with Key 2. On success, the key fob is authenticated to the car and bilateral authentication is finished.

2 Tracking

The protocol stack includes the "ReadUID" command to retrieve the 32-bit UID from the key fob. There is no security mechanism in place which would require authentication by the reader. Therefore, any reader can request the UID and the key fob can be potentially tracked via a number of readers installed at various places.

Tracking could be prevented if the UID is not returned in cleartext, but dependent on a shared secret and a nonce. A simple example is to use the existing AES encryption E_K with one of the pre-shared keys K in a tweakable block cipher construction $\tilde{E}_K$ [9].

$$\tilde{E}_K(\text{nonce}, \text{UID}) = E_K(\text{nonce} \oplus E_K(\text{UID})) \tag{1}$$

The result of $\tilde{E}_K$ will vary with the nonce and the UID will be protected even when the nonce is revealed. Thus, even though the key fob can be still queried by any reader, the result cannot be used any more to track it.

There are two options for the values returned by the key fob depending on the actual functional requirements. If the complete result of $\tilde{E}_K$ is returned alongside with the nonce, the reader can decrypt it and arrive at the original UID. Thus, the full functionality of the original "ReadUID" command is retained. This comes at the price of a relatively high communication overhead as the key fob needs to send the 128-bit ciphertext $\tilde{E}_K$ and the nonce. The computational overhead would essentially be the generation of the nonce and two AES encryptions on the key fob side and two AES decryptions on the reader side.

Alternatively, the reader could still check for a specific UID if only a part of the result of $\tilde{E}_K$ were returned with the nonce. This could be useful if the reader requires the "ReadUID" command exclusively to check for a specific UID. We denote this new command as "CheckUID" and its functionality is shown in Figure 2. It's advantage is a shorter response and a better response time of the key fob compared to the enhanced "ReadUID" command.

By varying the size of the nonce and the portion of $\tilde{E}_K$ to be checked (M-bit RespM), the security and communication overhead can be balanced. For example, using a 32-bit portion of $\tilde{E}_K$ for checking, a similar resilience against

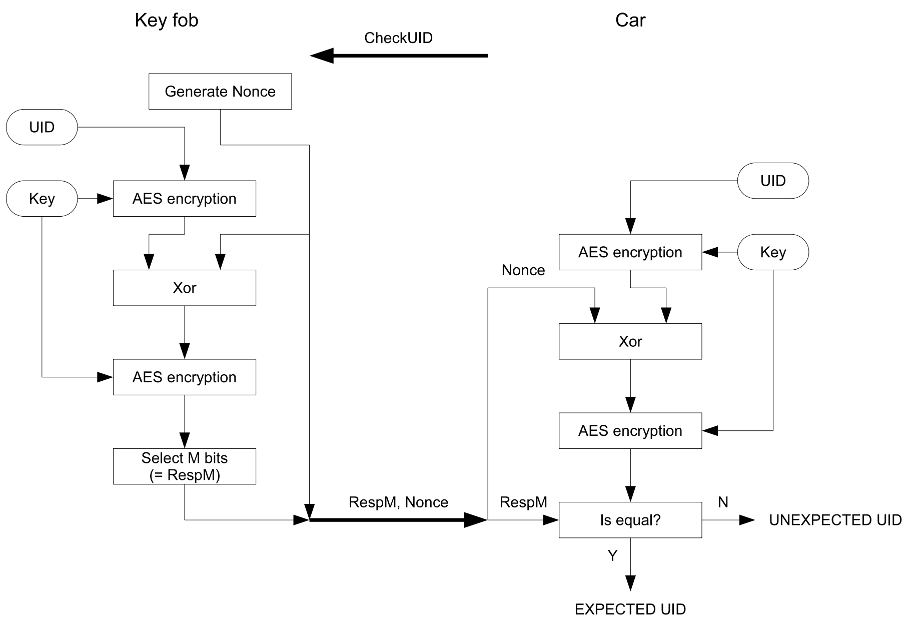


Fig. 2: Enhanced "CheckUID" command with resistance against tracking.

accidentally matching UIDs would be introduced as in the original protocol stack with 32-bit UIDs. The communication overhead would consist of the extra bits of the nonce and the computational overhead would be the generation of the nonce for the key fob and two extra AES encryptions for key fob and reader each. The encrypted UID (E_K(UID)) could also be pre-computed and stored which would reduce the computational overhead by one AES encryption for each side.

In both cases, the key fob must be able to generate nonces. This might require a key fob with slightly higher capabilities as set out in the protocol stack specification. Generation of nonces is also required by the countermeasure to the attack described in Section 5.

3 Denial-of-Service Attacks

The protocol stack includes commands for writing new cryptographic keys to the key fob, which replaces the old keys used for authentication. There are two different modes for doing this: In open mode, a "Learn Secret Key1" or "Learn Secret Key2" can be issued by any reader in order to set new keys. In secure mode, an encrypted key is sent by the reader device, decrypted by the key fob and the result is set as new key as shown in Figure 3. The key used for encrypting the new key is the so-called Default Secret Key which is factory set.

Overwriting keys in open mode is trivial, as the malicious reader only has to send the according command to set the keys to those of her choice. However,

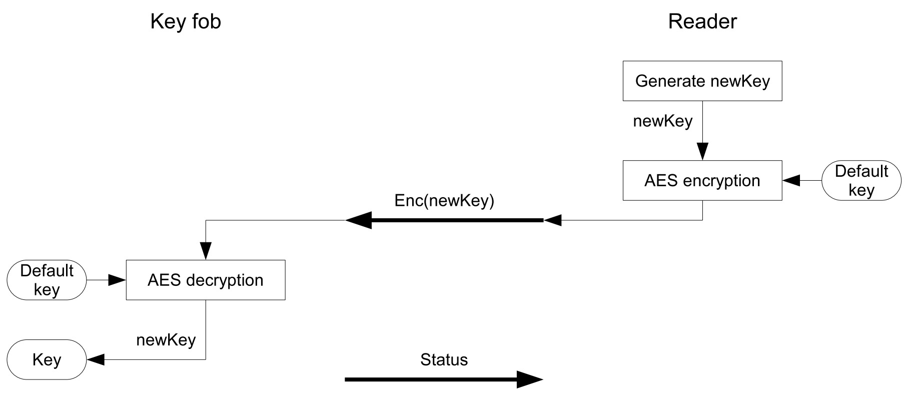


Fig. 3: LearnSecretKey command in secure mode.

even in secure mode it is possible to overwrite keys though the value of the new keys stays hidden to the attacker. This is possible because the secure key learn command only uses the encrypted key but no integrity check for it. Therefore, an attacker can send a random value as encrypted key and the key fob will set the decrypted value as new key.

Thus, in both open and secure mode, keys can be overwritten without the need of knowing a shared secret. Once this has been done, the key fob will no longer work with the car. If the key fob is queried in intervals while the car is in motion, it might even be possible to force the immobilizer to stop the car by overwriting the keys.

The open mode is vulnerable against this attack per design. To defend against the attack in secure mode, a message authentication code (MAC) should be included with the encrypted key and the key should only be overwritten when the MAC is verified successfully. This entails communication overhead for transmission of the MAC from the reader to the key fob and computational overhead of MAC generation in the reader and MAC verification in the key fob.

4 Relay Attack with Genuine Key Fob

Another type of attack tricks the car into thinking that the key fob is in its immediate vicinity when it is actually located further away. Such relay attacks have been known as early as 1976 [2] and have been practically demonstrated, e.g. in [4] for the EMV chip and PIN setting. In the current setting, this attack relays messages between the genuine key fob and the car through a transparent reader (close to the genuine key fob) connected to a transparent key fob (close to the car) as shown in Figure 4. Such an attack would require two cooperating attackers, one bringing the transparent reader close to the genuine key fob and the other gaining entry to the car and bringing the transparent key fob close to the car's reader.

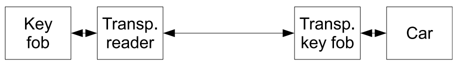


Fig. 4: Relay attack with transparent reader and key fob.

A potential countermeasure to this relay attack is to measure the communication delay between the reader's challenge and the key fob's response in order to detect the actual distance between the communicating endpoints. Alternatively, a dedicated protocol, like the distance bounding protocol used in [4] could be employed. However, the protocol stack includes a mechanism to defeat such countermeasures. If the transparent key fob fakes an uplink CRC error, this forces the car to send a "Repeat Last Response" command. The attacker can use the extra time for the repeated response to get the actual response from the genuine key fob.

This remote attack could be defended against with the measurement of the communication delay of the key fob by the car and by abandoning the mechanism of requesting a repeat of the the key fob's response in answer to a CRC error. Instead the whole sequence of commands and responses should be repeated when a CRC error is encountered. This gives the attacker no time to hide the extra communication delay introduced by the transparent reader and key fob. Measurement of the communication delay might require extra components (e.g. a high-precision oscillator) at the car's side.

5 Replay Attack on Authentication

A unique property of the bilateral authentication protocol in the immobilizer stack is that the key fob is not required to generate nonces. Instead, the encrypted nonce from the reader is "reused" as the challenge from the key fob. While this makes the structure of the key fob simpler, it also means the commands from the reader can be recorded and replayed at a later time to achieve authentication. Thus an attacker can pretend to be an authenticated reader, which gives her access to advanced commands like "Read User Memory" and "Write User Memory".

A defense against this attack is to have the key fob generate the challenges for the reader. Without a challenge from the key fob, the replay of the reader command will lead to a successful authentication of the reader.

6 Spoofing Attack on Memory Access Protection

The protocol stack allows the reader to lock the EEPROM sections AP1 to AP3 via a "Write Memory Access Protection" command. This command is accepted by the key fob without prior authentication. Depending on the actual use of these EEPROM sections, an attacker could impair the functionality of the key fob by locking them with a spoofed command.

By requiring prior authentication for the "Write Memory Access Protection" command this attack can be prevented.

7 Conclusions

In this paper we have identified a number of potential security vulnerabilities in an open car immobilizer stack. The vulnerabilities include tracking of key fobs, denial-of-service attacks to render key fobs useless, achieving key fob authentication despite absence of the key fob (relay attack), achieving reader authentication via a replay attack, and a spoof attack to lock out EEPROM sections of the key fob. For each of the identified vulnerabilities we propose countermeasures. This proves the great value of the openness of the protocol stack to public review. Some of our proposed countermeasures can be implemented rather easily while others require enhanced functionalities from the reader and/or the key fob.

Acknowledgements. The research described in this paper has been supported by EPSRC grant EP/H001689/1. The information in this document reflects only the author's views, is provided as is, and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. Atmel. Open Source Immobilizer Protocol Stack. Available online at `http://www.atmel.com/dyn/products/tools_card.asp?tool_id=17197` (registration required), 2010.
2. J. H. Conway. *On Numbers and Games*. Academic Press, 1976.
3. Daily RFID Co., limited. LF RFID Reader-03. `http://www.rfid-in-china.com/2008-09-06/products_detail_2140.html`.
4. S. Drimer and S. J. Murdoch. Keep Your Enemies Close: Distance Bounding Against Smartcard Relay Attacks. In *Proceedings of the 16th USENIX Security Symposium*, pages 87–102, 2007.
5. GAO RFID Inc. 125 kHz Long Range Reader. `http://www.gaorfid.com/index.php?main_page=product_info&products_id=363`.
6. F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. W. Schreur, and B. Jacobs. Dismantling MIFARE Classic. In S. Jajodia and J. Lopez, editors, *13th European Symposium on Research in Computer Security (ESORICS 2008), Malaga, Spain, 6-8 October, 2008, Proceedings (to appear)*, Lecture Notes in Computer Science. Springer Verlag, 2008.
7. S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on KeeLoq. In N. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.
8. P. Lepek. Configurable, Secure, Open Immobilizer Implementation. In *Proceedings of the 8th Embedded Security in Cars (ESCAR) Conference*.

9. M. Liskov, R. L. Rivest, and D. Wagner. Tweakable Block Ciphers. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 31–46. Springer, 2002.
10. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. Series on Discrete Mathematics and its Applications. CRC Press, 1997. ISBN 0-8493-8523-7, Available online at `http://www.cacr.math.uwaterloo.ca/hac/`.
11. National Institute of Standards and Technology. AES Competition Website (archived). `http://csrc.nist.gov/archive/aes/index.html`.
12. National Institute of Standards and Technology. SHA-3 Competition Website. `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`.

Extended Abstract: Markov Game Analysis for Attack-Defense of Power Networks

Chris Y. T. Ma$^\sharp$, David K. Y. Yau$^{\diamond\sharp}$, Xin Lou$^{\dagger\sharp}$, and Nageswara S. V. Rao‡

$^\sharp$ Advanced Digital Sciences Center, Illinois at Singapore
$^\diamond$ Purdue University, West Lafayette, IN, USA
† City University of Hong Kong, Hong Kong
‡ Oak Ridge National Laboratory, TN, USA

Abstract. Electricity grids are critical infrastructures. They are credible targets of active (e.g., terrorist) attacks since their disruption may lead to sizable losses economically and in human lives. It is thus crucial to develop decision support that can guide administrators in deploying defense resources for system security and reliability. Prior work on the defense of critical infrastructures has typically used static or Stackelberg games. These approaches view network interdictions as one-time events. However, infrastructure protection is also a continual process in which the defender and attacker interact to produce dynamic states affecting their best actions. In this paper, we use zero-sum Markov games to model these interactions subject to underlying uncertainties of real-world events and actions. We solve equilibrium mixed strategies of the players that maximize their respective minimum payoffs with a time-decayed metric. Using results for a 5-bus system [1] and a WCSS 9-bus system [2], we illustrate that our game model can provide useful insights.

1 Introduction

Electricity networks are critical infrastructures. Their disruptions can have severe economic, social, and security consequences. For example, lives may be endangered if power is lost for life saving procedures in hospitals. Loss of power may also prevent communication, stall work, cripple transportation, and/or lead to other major failures that can bring entire nations to a standstill. Because of their importance, power networks are credible targets for active (e.g., terrorist) attacks. On the other hand, protecting these networks is extremely challenging, due to their expansive geographical extents and complex interdependencies between system components. For example, transmission lines may run for miles in the open, and the system must maintain stable and prespecified power quality (e.g., frequency, voltage, and phase synchronization) for performance and safety of equipment.

To protect critical infrastructures from attacks, administrators need tools that support prudent decision making. In particular, administrators need to make informed decisions about where to deploy finite resources to harden a system for maximum resiliency against adversaries. Such guidance for infrastructure protection has been obtained using Markov decision processes (MDP) or game theory. In MDP [3], the system is modeled as a set of states with Markov transitions between them. The problem is to optimize the actions of a "player" (e.g., the defender) under probabilistic outcomes of these actions. The solution optimizes the actions of a single player only. It is suitable for a defender to maximize system reliability against passive disruptors of known probabilistic behavior. Collectively, these disruptors may represent "nature," which may disrupt components by indeliberate events such as bad weather or normal wear-and-tear.

Game theoretic approaches for infrastructure protection, on the other hand, postulate a strong attacker – one capable of devising its own best counter strategies against the defender. In a *static game* [4], both players choose their moves simultaneously. In another form of leader-follower Stackelberg games [5], the optimization of the players' strategies is a bilevel problem. At the inner level, the follower maximizes its payoff *given* a leader's strategy. At the outer level, the leader chooses a strategy S to maximize its own payoff subject to the follower's solution of the inner problem defined by S.

The above kinds of games view network interdictions as one time events. However, infrastructure protection is also a continual process in which the players interact to produce dynamic states affecting their respective best actions. Markov games model these interactions subject to inherent uncertainty in the underlying physical system. They can be viewed as generalizations of MDP to an adversarial setting. For the protection of power networks, we assume that the attacker deploys resources to disrupt transmission lines in a power grid,[1] and its goal is to maximize the amount of load shedding. The defender's goal, on the other hand, is to deploy defense resources to minimize the amount of load shedding in the face of such attacks. The directly opposing goals of the attacker and defender lead to a zero-sum game formulation naturally.

Game theoretic analysis has typically assumed a full information setting. In practical situations, information is a valuable asset having significant effects on achieved payoffs. Control and sensing information communicated in future smart grids may be a valuable source of information for would-be attackers. For example, advanced meter infrastructures (AMIs) may give a comprehensive view of the distribution of load and resulting power flows. More sophisticated attackers may even infer the types of load based on their power signatures [6], leading to knowledge about the cost functions in our model. The role of on-line information gives a cyber dimension of smart grid protection as a *cyber-physical system* problem.

Our **contributions** are as follows. (i) We model the attack-defense of power networks as a Markov game. We solve equilibrium mixed strategies of the players that maximize their respective minimum payoffs by a time-decayed metric under uncertainty. (ii) We show that after our algorithm converges, the solution in each state is equivalent to that of a static game with a composite payoff matrix. Analysis of this composite matrix simplifies the interpretation of results obtained by our algorithm. (iii) We apply our solution to two realistic power systems. We

[1] Transmission lines can be considered particularly vulnerable targets due to the impossibility of physical isolation. However, our problem can be readily generalized to consider other system components.

contrast our numerical results with those of static games, and show that their analysis leads to useful insights in sometimes subtle situations.

2 Related Work

MDP has been used to analyze the security and vulnerability of urban infrastructures. Jones *et al.* [7] use it to analyze the actions of an intruder into transportation facilities. Jha *et al.* [3] use MDP to interpret attack graphs in communication networks, so that a minimal set of security measures can be determined that will guarantee the safety of a system. Their work optimizes the actions of the defender against a passive attacker, whose strategy is fixed and given.

Game theory has been widely used to analyze the security of critical systems. The competition between a defender and an attacker in this context has been modeled as leader-follower Stackelberg games [5], [8], and static games [4], [9]. These games analyze one move of each player only, and so they treat network interdictions as one time events. In practice, the defender may interact with the attacker in repeated plays that evolve the system state dynamically. Alpcan *et al.* model these repeated plays under uncertainty as a Markov game. They use the game model to design an intrusion detection system for a communication network [10], and compare their results with those obtained using static games.

3 Problem Formulation

In a power grid, generators supply electricity and loads consume it. They are attached to a set of buses – which we call generation and load buses, respectively – interconnected by a network of transmission lines of given capacities. Henceforth, we refer to transmission lines as *links.*

An attacker aims to disrupt the power network by bringing down one or more links, in order to cause maximum “disruption” of the load. A defender aims to minimize this disruption. It does so by reinforcing links that are up, and repairing links that are down. In a baseline case, disruption is measured simply as the amount of load (in power unit) that must be shed due to the link failures. More generally, shedding different loads may have different adverse impact which we call *cost.* A *cost function* for a load bus, say l, is given by $u_l(x, y)$, which specifies the cost of reducing the load from x to y (in power units) on l. In this case, disruption is measured as the total cost of shed load due to the link failures.

We define a Markov game as follows. The state of the game refers to the set of links that are currently up (links that are not up are down) in the power network. The game proceeds in discrete time steps. In each time step, the players choose a pair of actions which, together with underlying probabilistic physical events, may cause state transitions in a Markov manner. For the attacker, the action is the link that it chooses to attack. For the defender, the action is the (down) link that it chooses to repair or the (up) link that it chooses to reinforce. The players have limited budgets in that in each time step, the attacker (respectively defender) can choose a limited number of links to attack (respectively repair/reinforce) only.

We use the following notations throughout the paper.

- A_p: Action set of player p, where $p = a, d$, corresponding to the attacker and defender, respectively.
- S: Set of game states, where each state is an enumeration of the status of the links in order. We use "u" and "d" to denote the up or down status, respectively.
- $PD(A)$: Set of mixed strategies over the action set A.
- p_{pf}: Probability for an up link to fail in a time step upon attack, when it is reinforced by the defender in that time step.
- p_{upf}: Probability for an up link to fail upon attack, when it is not reinforced. We have $0 \leq p_{pf} \leq p_{upf} \leq 1$.
- p_{pr}: Probability for a down link to recover (i.e., become up) in a time step, when it is repaired by the defender and not attacked by the attacker in that time step.
- p_{upr}: Probability for a down link to recover when it is not repaired by the defender and not attacked by the attacker. We have $0 \leq p_{upr} \leq p_{pr} \leq 1$.

We assume that the attacker can attack a link that is already down. Such an action will reduce the probability that the link recovers. For example, if a down link is repaired by the defender and further attacked by the attacker in a time step, then its probability of recovery is $p_{pr} \times (1 - p_{upf})$. If the down link is not repaired by the defender, then its probability of recovery under attack is $p_{upr} \times (1 - p_{upf})$.

Since the load shedding goals of the attacker and defender are directly opposing, we have a zero-sum game. A pair of player actions in a state will bring an *immediate reward* for the players. For the attacker, this reward is the expected cost of shed load due to the resulting probabilistic transitions to the possible next states. The defender's immediate reward is the negative of this number. Further to the immediate reward, each possible state transition if realized will bring the game to a new state, where the game will carry on. A further immediate reward will be obtained in the new state with further new state transitions, and so on. Hence, a pair of actions taken in a state will accrue a *long-term reward* in general.

Formally, define $R(s, a, d)$ as the expected immediate reward for the attacker when it takes action a and the defender takes action d in state s. (Reward for the defender is the negative of this number.) Further define $Q(s, a, d)$ as the expected long-term reward for the attacker when it takes action a and the defender takes action d in state s. (Expected long-term reward for the defender is the negative of this number.) The *value* of state $s \in S$ for the attacker in the Markov game is

$$V_a(s) = \max_{\pi \in PD(A_a)} \min_{d \in A_d} \sum_{a \in A_a} Q(s, a, d)\pi_a, \tag{1}$$

where π_a is the probability of action a in the optimal mixed strategy π of the attacker. The expected long-term reward, *quality*, of action a against action d in state s is

$$Q(s, a, d) = R(s, a, d) + \gamma \sum_{s'} T(s, a, d, s')V_a(s'), \tag{2}$$

where $T(s,a,d,s')$ is the state transition $T : S \times A_a \times A_d \to S$, and γ is a discount factor satisfying $0 \leq \gamma < 1$. γ gives the discount factor of future rewards on the optimal decision. Small values of γ emphasize near-term gains while large values emphasize future rewards. γ may also be interpreted as the belief of possible future interactions held by the players.

Similarly, the *value* of state $s \in S$ for the defender is

$$V_d(s) = \min_{\pi \in PD(A_d)} \max_{a \in A_a} \sum_{d \in A_d} Q(s,a,d)\pi_d. \tag{3}$$

Notice that in general, $V_a(s)$ and $V_d(s)$ computed from Eq. 1 and Eq. 3 are different. In particular, $V_a(s) \leq V_d(s)$, where Eq. 1 corresponds to the primal problem and Eq. 3 corresponds to the dual problem. The inequality expresses weak duality relating the primal and dual problems in general [11, Section 5.4]. When the Markov game is zero-sum, however, strong duality applies [11, Section 5.4] and equality holds due to the *strong max-min property.* Hence, we use $V(s)$ to denote the value of state $s \in S$, and $V(s) = V_a(s) = V_d(s)$. The optimal solutions computed individually by the two players are therefore best responses to each other and they are in Nash equilibrium. The equilibrium solutions are necessarily Pareto-optimal, because we cannot improve the payoff of one player without hurting that of the other in a zero-sum game.

4 Markov Game Solution

We now solve the Markov game defined in Sec. 3. Our goal is to compute equilibrium best *policies* for both players, where a policy is the set of per-state optimal mixed strategies of the player concerned, and an optimal strategy is one that maximizes the minimum long-term reward under the best strategy of the opponent. It is known that every Markov game has a non-empty set of optimal policies for each player, and one of them is *stationary*, i.e., it is time-independent [12]. Our solution will find this optimal stationary policy for each player. Once the optimal policies of the players are determined, the Markov transition probabilities are completely defined and the system will evolve as a standard Markov process.

We consider the case in which both players have complete information about the game. The solution is a generalization of *value iteration*, a common dynamic programming technique for solving MDPs [12], [10], to a game-theoretic setting.

Recall from Sec. 3 that the value of state $s \in S$ in the game is given by Eq. 1 for the attacker, and by Eq. 3 for the defender. The optimal mixed strategy π of the attacker can be obtained by solving the following linear program:

$$\begin{aligned} &\max_{\pi \in PD(A_a)} V(s), \\ \text{s.t.} \quad &\sum_{a \in A_a} Q(s,a,d)\pi_a \geq V(s), \\ &\sum_{a \in A_a} \pi_a = 1, \\ &\pi_a \geq 0. \end{aligned}$$

```
1. Set $V(s) = 0$ for all $s \in S$
2. repeat
3.     for all $s \in S$ and $a \in A_a$ and $d \in A_d$ do
4.         Update $Q$ according to Eq. 2
5.     end for
6.     for all $s \in S$ do
7.         Update $V$ according to Eq. 1
8.     end for
9. until $V(s) \to V^*$, i.e., $V(s)$ converges.
```

Fig. 1. Dynamic programming algorithm for solving the Markov game.

		Link attacked by attacker				
		1	2	3	4	5
Link protected by defender	1	61.05	0	112.1	472.1	230
	2	122.1	0	112.1	472.1	230
	3	122.1	0	56.05	472.1	230
	4	122.1	0	112.1	236.1	230
	5	122.1	0	112.1	472.1	115

(a)

		Link attacked by attacker				
		1	2	3	4	5
Link protected by defender	1	210.6	152.4	279.7	674.4	445.5
	2	292.4	140.6	279.7	674.4	445.5
	3	292.4	152.4	204.2	674.4	445.5
	4	292.4	152.4	279.7	401.6	445.5
	5	292.4	152.4	279.7	674.4	287.2

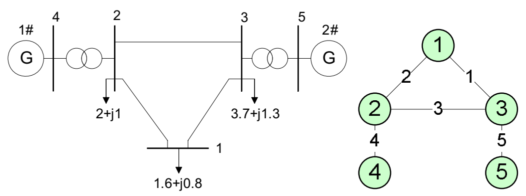
(b)

Table 1. Quality of actions of the two players in state {u,u,u,u,u} for (a) a static game that does not consider future rewards, (b) the full Markov game. Numbers are payoffs for the attacker. Hence, the attacker prefers larger numbers, while the defender prefers smaller numbers. $p_{pf} = 0.5, p_{upf} = 1, p_{pr} = 0.6, p_{upr} = 0$.

The optimal π of the defender can be obtained by the above formulation with the order of the maximization and minimization swapped.

The value iteration algorithm to compute the optimal Q and V for given s, a, d is specified in Fig. 1. The algorithm iteratively estimates the values of V and Q by treating the equal signs in Eqs. 2 and 1 as assignment operators for updating the estimates. These estimates will converge to their correct values [13]. Notice that each iteration of the algorithm produces a mixed strategy for one player in state s by linear programming (Line 7). These mixed strategies will similarly converge to the optimal one, and hence we obtain one player's optimal policy when the algorithm terminates. We then use the converged Q's to solve for V's by linear programming from the perspective of the other player, and obtain the optimal policy of the other player.

Notice that we initialize $V(s) = 0$. As a result, the mixed strategy of the player after the first iteration is its optimal mixed strategy in a *static* game that does not consider rewards in future time steps, and the obtained $V(s)$ corresponds to the payoff in state s of this static game. For instance, for the 5-bus system shown in Fig. 2, Table 1(a) shows the payoff matrix of the static game for state {u,u,u,u,u}. Notice that the matrix shows the payoff to an attacker, and hence, the attacker prefers an action that returns a larger number, while the defender prefers an action that returns a smaller number. As we consider future rewards, the payoff matrix will evolve during the iterative process of the algorithm. When their effects are fully considered, Table 1(b) shows the "composite" payoff matrix for the same state {u,u,u,u,u} after the convergence of V in Line 9. For any state s, the optimal mixed strategy of the player in the Markov game is equivalent to the optimal mixed strategy solved for an equivalent static game with the composite matrix as payoffs. This view facilitates the interpretation of results obtained for the Markov game.



(a) Bus diagram (b) Link diagram

Fig. 2. 5-bus system.

Bus ID	1	2	3	4	5
Load (MW)	160	200	370	0	0
Supply (MW)	0	0	0	500	257.8

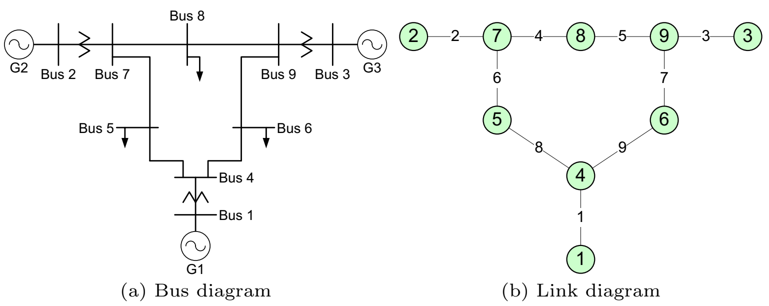
Table 2. Load and supply distribution of 5-bus system.

5 Evaluations

We present numerical results to illustrate solutions of the Markov games, which include the static games as a special case ($\gamma = 0$), using the failure and recovery probabilities as follows, $p_{pf} = 0.5, p_{upf} = 1, p_{pr} = 0.6, p_{upr} = 0$, unless stated otherwise. We assume that both players have complete information of the game. The cost function of load shedding is the amount of load shed. We have results for a 5-bus system [1] and a WCSS 9-bus system [2]. Their bus and link diagrams are given in Figures 2 and 3, respectively, and their per-bus aggregate generation and load are listed in Tables 2 and 3, respectively. We will focus on the 5-bus system for illustration of the more detailed results, since its relative simplicity facilitates the exposition.

Notice that certain links in a power system are particularly important, in that interdicting such a link by itself will already cut off a large amount of power flow from generation to load. In the 5-bus system, links l_4 and l_5 are particularly important, with l_4 being more so. In the 9-bus system, links l_1, l_2, and l_3 are particularly important, with l_2 being the most. These important links usually form the focus of the player strategies.

Fig. 4 shows the player strategies in selected states of the Markov game for the 5-bus system. In the figure, a bar labeled $p_a(x)$ gives the probability that the attacker will attack link x, and a bar labeled $p_d(x)$ gives the probability that the defender will repair link x (if x is down) or reinforce link x (if x is up).



(a) Bus diagram (b) Link diagram

Fig. 3. Standard WCSS 9-bus system.

Bus ID	1	2	3	4	5	6	7	8	9
Load (MW)	0	0	0	0	125	90	0	100	0
Supply (MW)	71.64	163	85	0	0	0	0	0	0

Table 3. Load and supply distribution of WCSS 9-bus system.

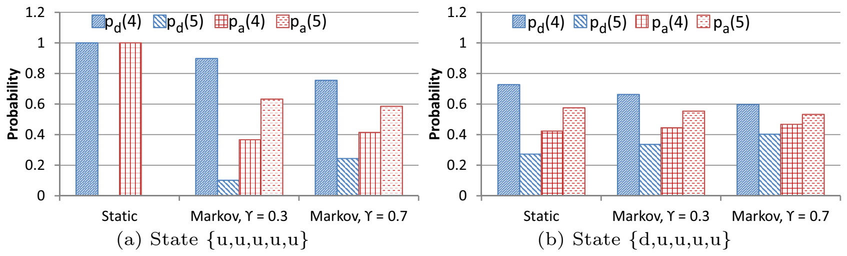


Fig. 4. Player strategies in selected states of the Markov game for the 5-bus system. Both players have budgets to affect one link only in a time step.

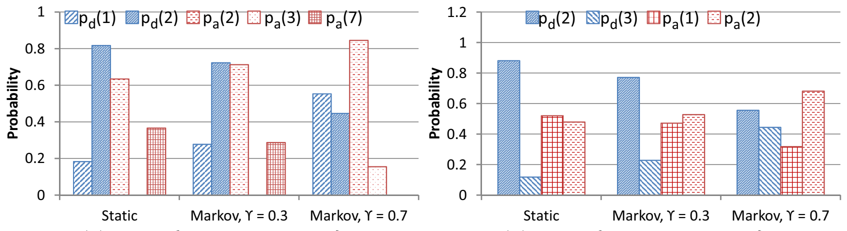
For example, $p_a(5)$ represents the probability for the attacker to attack l_5. Only actions with non-zero probabilities are included in the figure. The defender and the attacker have budgets to affect one link only in a time step. The results show that the optimal policies of the players may change significantly as we vary γ from zero (static game) to 0.7.

For instance, Fig. 4(a) shows that in state {u,u,u,u,u}, the defender progressively shifts its focus from reinforcing l_4 to reinforcing l_5, while the attacker also attacks l_5 apart from l_4, as γ increases. This observation can be explained using the payoff matrix of the static game (Table 1(a)) and the composite payoff matrix of the Markov game when $\gamma = 0.3$ (Table 1(b)). Notice that the numbers shown are the costs of load shedding and hence represent payoffs for the attacker – the attacker prefers higher numbers while the defender prefers lower numbers. Table 1(a) shows that in the static game, the payoff of attacking l_4 is always higher than that of attacking l_5, i.e., both l_4 and l_5 are important but l_4 is even more so. Hence, the attacker will only attack l_4, and the defender will always defend l_4 to minimize its cost. However, Table 1(b) shows that in the Markov game, the payoff of attacking l_4 is always higher than attacking l_5, *except* in the case that the defender is also reinforcing l_4. Hence, when l_4 is being reinforced with sufficiently high probability, the attacker begins to use a mixed strategy that includes l_5. This illustrates a subtle interplay between the players: Although a *successful* attack on l_4 will bring higher benefit for the attacker, it is also more difficult if l_4 is also reinforced by the defender. Hence, the attacker shifts some of its focus to the easier target l_5 since that link is also important.

Fig. 5 shows selected strategies of the players in the Markov game for the 9-bus system.

6 Conclusion

We have presented a Markov game analysis of attack-defense in power systems. Our results complement related results using static games or Stackelberg games. We show that consideration of repeated plays under Markov-type uncertainties will in general modify the strategies of the players relative to games with single



(a) State {d,u,u,u,u,u,u,u,u} (b) State {u,u,d,u,d,u,u,u,u}

Fig. 5. Selected player strategies for the 9-bus system. Both players have budgets to affect one link in a time step. $p_{pf} = 0.5, p_{upf} = 1, p_{pr} = 0.6, p_{upr} = 0$.

plays. This is because the players will need to consider the impact of a current action on the future plays, although the future rewards are generally discounted by a factor γ. We have applied our analysis to a 5-bus system that has been studied in the literature and a WCSS 9-bus system. The relative simplicity of the 5-bus system has allowed us to analyze its results in detail. Our analysis exposes subtle features of the game solutions, considering the *values* of different game states to the players and the intricate interplay between their strategies. It is also interesting to apply our analysis to other critical infrastructures.

References

1. : Calculation of The Electrical Power System. Hydro-electricity Press (1978)
2. Anderson, P.M., Fouad, A.A.: Power System Control and Stability. Galgotia (1981)
3. Jha, S., Sheyner, O., Wing, J.: Two formal analysis of attack graphs. In: Proc. of the IEEE workshop on Computer Security Foundations. (2002)
4. Holmgren, A., Jenelius, E., Westin, J.: Evaluating strategies for defending electric power networks against antagonistic attacks. IEEE Trans. Power Syst **22**(1) (2007)
5. Salmeron, J., Wood, K., Baldick, R.: Analysis of electric grid security under terrorist threat. IEEE Trans. Power Syst **19**(2) (2004)
6. Laughman, C., Lee, K., Cox, R., Shaw, S., Leeb, S., Norford, L., Armstrong, P.: Power signature analysis. IEEE Power & Energy Magazine **1**(2) (2003)
7. Jones, D.A., Davis, C.E., Turnquist, M.A., Nozick, L.K.: Physical security and vulnerability modeling for infrastructure facilities. In: Proc. of the Hawaii International Conference on System Sciences. (2006)
8. Brown, G., Carlyle, M., Salmeron, J., Wood, K.: Defending critical infrastructure. Interfaces **36**(6) (2006)
9. Chen, G., Dong, Z.Y., Hill, D.J., Xue, Y.S.: Exploring reliable strategies for defending power systems against targeted attacks. IEEE Trans. Power Syst **26**(3) (2011)
10. Alpcan, T., Basar, T.: Network Security: A Decision and Game Theoretic Approach. Cambridge University Press (2010)
11. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
12. Littman, M.: Markov games as a framework for multi-agent reinforcement learning. In: Proc. of the International Conference on Machine Learning. (1994)
13. Owen, G.: Game Theory: Second edition. Academic Press (1982)

SR-ORAM: Single Round-trip Oblivious RAM

Peter Williams and Radu Sion

{petertw, sion}@cs.stonybrook.edu
Stony Brook University

Abstract. We present the first single-round-trip polylogarithmic time Oblivious RAM requiring only logarithmic client storage. Taking only a single round trip to perform a query, SR-ORAM has a communication/computation cost of $O(\log n)$, with $O(\log^2 n \log\log n)$, and under 2 round trips, overall amortized per-query communication requirements. The trusted client folds an entire interactive sequence of Oblivious RAM requests into a single query object that the server can unlock incrementally, to satisfy a query without learning its result. This results in an Oblivious RAM secure against an actively malicious adversary, with unprecedented speeds in accessing large data sets over high-latency links.

1 Introduction

Oblivious RAM (ORAM) allows a client to read and write data hosted by an untrusted party, while hiding both the data and the *access pattern* from this untrusted host. Access pattern privacy is a critical component of data privacy. Without access pattern privacy, the act of reading and writing remote data leaks potentially essential information about the data itself, making it impossible to achieve full data confidentiality. Since the introduction of the first Oblivious RAM in [6], approaches to increase query throughput have been relentlessly sought. Nevertheless, and despite the wide range of potential applications, practical Oblivious RAM constructions have remained elusive until very recently.

One of the most significant challenges to providing practical ORAM is that these interactive protocols require a large number of client-server round trips, resulting in large, often impractical, online query latencies. For example, [7] requires $\log_2 n$ round trips, translating to an online cost alone of over 1200-1500ms per query on a 1 terabyte database (e.g., for 10KB blocks), assuming a network link with a latency of just 50ms.

This paper provides a simple and direct solution to the challenge: SR-ORAM, a single-round-trip ORAM. SR-ORAM requires a single message to be sent from the client to the server and thus incurs a single round-trip (for a total online cost of 50ms in the example above). Moreover, SR-ORAM does not greatly affect the offline, amortized cost.

The basic idea behind SR-ORAM is to fold the interactive queries into a single non-interactive request without sacrificing privacy. The client constructs a set of values (a "query object") that allows the server to selectively decrypt pieces, depending on new values obtained during its traversal of the database. Each component of the query object unlocks only a specific single new component—which allows server database traversal progress while preventing it from learning anything about the overall success of the query.

Our construction is based on the Bloom filter ORAM of [14], since it lends itself conveniently to use of a non-interactive query object and provides defenses against actively malicious adversaries (not only curious). We also make use of the randomized shell sort defined in [8], since it allows the more stringent client storage requirements of SR-ORAM (when compared to [14]).

Other ORAMs with constant numbers of round trips exist; Section 3 reviews recent solutions. However, SR-ORAM is the first to provide a constant-round-trip *polylogarithmic time* construction that assumes only *logarithmic client storage.*

2 Model

A capacity-constrained *client* desires to outsource storage to an untrusted party (the *server*). The client has enough local non-volatile storage to manage keys and certificates, plus enough volatile RAM to run the ORAM client software (logarithmic in the size of the outsourced data). Moreover, since the client reads and writes sensitive data, it needs to hide both the data content and access pattern. Thus, the client needs low-latency, private access to this remote disk.

Data is accessed in "blocks", a term used to denote a fixed-size record. "Block" is used instead of "word" to convey target applications broader than memory access (file system and database outsourcing, in particular, seem to be lucrative targets). Block IDs can be arbitrary bit sequences.

Participants

Communication between the user and the ORAM Client is secured, e.g., with access controls on inter-process-communication if they are on the same machine, or with SSL otherwise. Communication between the ORAM Client and ORAM Server is also secured, e.g., with a transport-layer protocol such as SSL.

ORAM Client: The trusted party providing the following (self-explanatory) interface to the user: read(id): val; write(id, val). The Client-Server protocol details are implementation-specific (and typically optimized to the instance to minimize network traffic and the number of round trips).

The client keeps track of two values between queries: its secret key and the current access count. From this, the current level keys, and the reshuffle count of each level, can be derived.

ORAM Server: the untrusted party providing the storage backend, filling requests from the instance.

Security Definitions

We will assume, and defend against, curious and potentially malicious (not constrained to follow the protocol) polynomially-bounded adversary in the random oracle model. The actively malicious defense is inherited from the underlying ORAM of [14].

For simplicity, timing attacks are not discussed here. Defenses include the introduction of client-side delays to uniformize query times—which can be done without affecting overall protocol complexity. Additionally, SR-ORAM assumes semantically secure symmetric encryption primitives and secure hash functions.

Notation

Throughout the paper, n refers to the database size, in blocks. The client secret key is sk. The number of times a given level has been shuffled (i.e. reconstructed) is called the "generation," and is abbreviated as gen. Oblivious Random Access Memory is ORAM; Bloom Filter is BF. Key size and hash function output size are both assumed to be c_0; c_1 is the Bloom filter security parameter.

To represent computational and communication costs in a comparable manner, complexities are represented in words, not bits. It is assumed that each word can hold an entire identifier, e.g., $O(\log n)$ bits.

3 Background

We start with an review of ORAM, and in particular, Bloom-filter-based ORAMs. We next review the highly-interactive Bloom-filter-based ORAM [14], which provides a convenient construction to build SR-ORAM from. Finally, we look at recent approaches to reduce the round trip cost.

3.1 ORAM Overview

Oblivious RAM [6] provides access pattern privacy to *a single client* (or software process) accessing a remote database (or RAM), requiring only logarithmic storage at the client. The amortized communication and computational complexities are $O(\log^3 n)$ for a database sized n.

In ORAM, the server-hosted database is a set of n semantically-secure encrypted blocks (with a *secret key* held by the client). Supported operations are read(id), and write(id, $newvalue$). The data is organized into $\log_2(n)$ *levels*, as a pyramid. Level i consists of up to 2^i blocks; each block is assigned to one of the 2^i buckets at this level as determined by a hash function.[1] Due to hash collisions each bucket may contain from 0 to $O(\log n)$ blocks. [2]

ORAM Reads. To obtain the value of block id, a client must perform a read query in a manner that maintains two invariants: (i) it never reveals which level the desired block is at, and (ii) it never looks twice in the same spot for the same block. To maintain (i), the client always scans a single bucket in every level, starting at the top (Level 0, 1 bucket) and working down. The hash function informs the client of the candidate bucket at each level, which the client then scans. *Once the client has found the desired block, the client still proceeds to each lower level, scanning random buckets instead of those indicated by their hash function.* For (ii), once all levels have been queried, the client re-encrypts the query result with the secret key and a different nonce (so it looks different to the server) and places it in the *top* level. This ensures that when it repeats a

[1] $\log_4(n)$ levels sized 4^i in the original, but for simplicty we use a branch factor of 2.

[2] This was originally specified as $\log n$ blocks, with a non-negligible probablity of bucket overflow, in which case a new hash function is tried. It was later shown (e.g., in [10]) that this results in an information leak.

search for this block, it will locate the block immediately (in a different location), and the rest of the search pattern is randomized. The top level quickly fills up; how to dump the top level into the one below is described later.

ORAM Writes. Writes are performed identically to reads in terms of the data traversal pattern, with the exception that the new value is inserted into the top level at the end. Inserts are performed identically to writes, since no old value will be discovered in the query phase. Note that semantic security properties of the re-encryption function ensure the server is unable to distinguish between reads, writes, and inserts, since the access patterns are indistinguishable.

Level Overflow. Once a level is full, it is emptied into the level below. This second level is then re-encrypted and re-ordered, according to a new hash function. Thus, accesses to this new *generation* of the second level will hence-forth be completely independent of any previous accesses. Each level overflows once the level above it has been emptied twice. The resulting re-ordering must be performed obliviously: once complete, the adversary must be unable to make any correlation between the old block locations and the new locations. A sorting network (e.g., [1] or [8]) is used to re-order the blocks thusly.

To enforce invariant (i), note also that all buckets must contain the same number of blocks. For example, if the bucket scanned at a particular level has no blocks in it, then the adversary would be able to determine that the desired block was *not* at that level. Therefore, each re-order process fills all partially empty buckets to the top with *fake* blocks. Recall that since every block is encrypted with a semantically secure encryption function, the adversary cannot distinguish between fake and real blocks.

3.2 Bloom filters

Bloom filters [3] offer a compact representation of a set of data items. They allow for relatively fast set inclusion tests. Bloom filters are *one-way*, in that, the "contained" set items cannot be enumerated easily (unless they are drawn from a finite, small space). Succinctly, a Bloom filter can be viewed as a string of b bits, initially all set to 0. To *insert* a certain element x, the filter sets to 1 the bit values at index positions $H_1(x), H_2(x), \ldots, H_k(x)$, where $H_1, H_2, \ldots, H_k$ are a set of k crypto-hashes. Testing set inclusion for a value y is done by checking that the bits for *all* bit positions $H_1(y), H_2(y), \ldots, H_k(y)$ are set.

By construction, Bloom filters feature a controllable rate of false positives r for set inclusion tests—this rate depends on the input data set size z, the size of the filter b and the number of cryptographic hash functions k deployed in its construction: $r = \left(1 - (1 - 1/b)^{kz}\right)^k$.

As will be seen below, the SR-ORAM Bloom filters are constrained by two important considerations. First, we need to minimize k, since this determines directly the number of disk reads required per lookup. Second, we need to guarantee that with high probability, there will be no false positives; i.e., r must be negligible to prevent a privacy leak, since a false positive reveals lookup failure to the curious server.

Encrypted Bloom Filters. The idea behind remotely-stored encrypted Bloom filters is to store their bit representation encrypted while still allowing client-driven Bloom filter lookups. This can be achieved, e.g., by storing the Bloom filters as bit strings XORed with client-side PRNG-driven key strings, or with individual bits stored and encrypted separately with semantic security (at the expense of additional storage).

Again, as will be shown, in SR-ORAM, instead of storing an encrypted bit for each position of the Bloom filter, we store part of a decryption *key*. Since the server cannot distinguish between the keys for bit-values of *1* and keys for bit-values of *0*, we retain the property that the server does not learn the success of the Bloom filter lookup.

3.3 Bloom filter-based ORAMs

The main contribution of [14] is the separation of level membership testing from item storage. Instead of checking for an item at a given level by reading the entire relevant bucket of $O(\log n)$-blocks, an encrypted Bloom filter is queried first. This indicates to the client which of two potential items (the real, if there, or a specific fake, otherwise) to retrieve. This saves a factor of $O(\log n)$ server storage while simultaneously speeding up level construction and querying.

More specifically, item location is encoded via a Bloom filter; any given item has membership in one of $\log_2 n$ Bloom filters, corresponding to one for each level. The level corresponding to the Bloom filter that contains this item is the level where the item must be retrieved from. Bloom filters are queried from the top down; maintaining access privacy requires that any given lookup be performed only once on any given Bloom filter. Once an item is found at a particular level, it is copied up to the top, so the request will be satisfied at a higher level next time. Random lookups will be performed on those levels and Bloom filters below where the item is found.

This is an interactive process requiring $\log_2 n$ round trips: the client needs to know the success of a lookup at a given level before it can start the query at the next level. Figure 1 illustrates this process of querying.

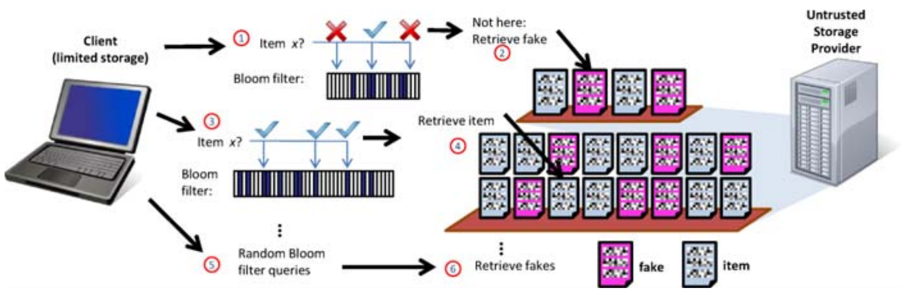


Fig. 1. Interactive Bloom filter querying. Both the lower level Bloom filter lookups and item lookups are dependent on the Bloom filter results of the levels above.

It is shown in [11] that the security analysis of [14] is incomplete, suggesting larger Bloom filters are needed to obtain negligible false positive rates. They also recommend a different selection in the tradeoff between Bloom filter size (affecting server storage and shuffle cost), and the number of hash functions chosen (affecting online cost). This adds a factor of $\log \log n$ to the Bloom filter construction cost. We apply these insights in the choices of Bloom filter parameters (number of hash functions k, and size in bits) and in the performance analysis (Section 7) of SR-ORAM.

We also note that [14] assumes a significant amount of temporary client storage necessary in the reshuffle step. This assumption is not suitable for our model. Instead, SR-ORAM uses an oblivious randomized shell sort [8] to support the level reshuffle and construct Bloom filters obliviously without client storage. This reduction in client storage requirements comes with performance penalties, as will be discussed later.

3.4 Other constant-round-trip ORAMs

Other recent approaches provide ways around the penalty of highly interactive protocols, at the cost of additional hardware or overwhelming requirements of client storage. The main issue in constructing a single round trip ORAM is that a request for an item depends on how recently an item was accessed. Maintaining this information at the client requires storage at least linear in the size of the outsourced database. Moreover, retrieving this information privately from the server is almost as difficult as providing ORAM. [3]

Secure Hardware. Secure hardware such as the IBM 4764 [9] can be placed server-side, using remote attestation to retain security guarantees for clients [2]. Secure hardware is typically an order of magnitude more expensive than standard processors. Due to heat dissipation difficulties it is typically also an order of magnitude slower. Moreover, the necessity of physical security to provide any guarantees makes such solutions vulnerable to a different class of attacks.

Constant-round-trip protocols using client storage. [12] maintains item location information at the client. Although at the outset, $n \log_2 n$ bits of client storage seems like a big assumption, the authors argue this is reasonable in some situations, since the block size is typically larger than $\log n$. They show that in practice, the local required client storage in practice is only a small fraction of the total database size. The recursive construction, using a second ORAM to store this level membership information, however, is interactive. SR-ORAM requires only $O(\log_2 n)$ bits of client storage (Section 5).

The non-interactive cache-based ORAM in [13] relies on k client storage to provide an amortized overhead of $O(n/k)$. The idea is to add previously unseen items to a cache, which gets shuffled back into the remote database when it fills. The high client storage requirements (and poor storage/performance tradeoff) make it unsuitable for our model. This idea is revisited under different assumptions in [4], with security formalization, but still requiring client storage.

[3] With the difference that this recursive ORAM only requires storing $O(\log \log n)$ bits per item, which is enough location information about the item to build the query.

A large number of *interactive* ORAM solutions have been proposed. An exhaustive review [5] is out of scope here; a full review should also include recent interactive *de-amortized* ORAMs. These resolve another drawback of many ORAMs (SR-ORAM included), the disparity between average-case and worst-case query cost.

4 A First Pass

This strawman construction modifies the Bloom filter ORAM of [14]. It has the structure, but not yet the performance, of the SR-ORAM construction. As detailed in Section 3.3, that Bloom filter ORAM uses encrypted Bloom filters to store level membership of items. To seek an item, the querying client must request a known fake item from each level, except from the level containing this item: the item is requested here instead. Which level the item is at depends only on how recently this item was last accessed. Since the client does not have storage to keep track of that, it checks the Bloom filters one at a time to learn if the item is at each level.

Moreover, since the main principle of level-based ORAMs requires *each item be sought once per level instance*, it is unsafe to query the Bloom filters past the level where this item is present. This explains why the checks must be interactive: once the item is found at level i, further accesses at the levels below ($i + 1$ through $\log_2 n$) entail only random Bloom filter queries corresponding to fake item requests. Then, putting the found item back at the top of the pyramid guarantees that later, it will be sought and found elsewhere, since the only way it gets back down to the lower levels is by riding a wave of level reshuffles.

We now describe how to safely turn this into a non-interactive process. Observe that in an interactive ORAM, if the client is requesting a recently accessed item j that happens to be in level 2, the access sequence will proceed as follows. This example is also illustrated in Figure 1. We use j to denote the item identifier, and sk for the secret key, and gen to represent the current generation of that level (a function of the total, global number of accesses, $accesscount$).

1. The client checks the level 1 Bloom filter for the item: reading the positions generated by Hash(sk | $level = 1$ | gen | j)
2. Upon seeing Encrypt(*0*) at one or more of those positions in the Bloom filter, the client learns the item is not at level 1. So it asks for a fake item instead, that is labeled as Hash(sk | $level = 1$ | gen | “fake” | $accesscount$)
3. The client now checks the level 2 Bloom filter for the item: reading the positions indicated by Hash(sk | $level = 2$ | gen | j)
4. Seeing Encrypt(*1*) at all of those positions, the client learns the item is at this level. This means it is safe to request the item here; the client asks for the block labeled Hash(sk | $level = 2$ | gen | “block” | $accesscount$)
5. Having already found the item, to maintain appearances and not reveal this fact to the server, the client continues to issue random Bloom filter lookups at each level i below. At each level it requests the fake blocks labeled Hash(sk | $level$ | gen | “fake” | $accesscount$)

Note that there are only $\log_2 n$ possible such access sequences, based on which level the item is found at (Figure 2). Each path starts with a real query. Real queries continue until an item is found, at which point only fake queries are issued from there on down. This limited number of possible sequences makes non-interactive querying possible.

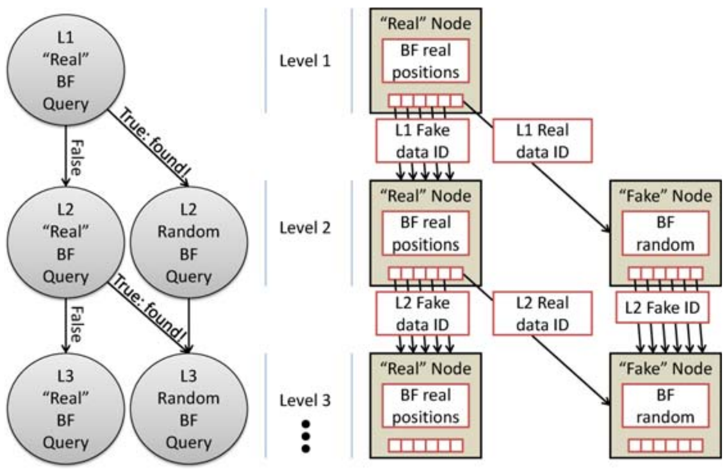


Fig. 2. Left: query paths. The client does not know at the time of query object construction which of the $\log n$ possible paths will be taken: it depends on where the data is ultimately found. Right: the query object. The server learns the edges corresponding to exactly one path. The gray shaded nodes contain the Bloom filter positions to read and a set of encrypted messages. The server will be able to decrypt one such edge at each level, revealing the data ID, to retrieve and include in the response to the client, and decrypting a node of the query object in the level below.

Since we have a finite number of these paths, our goal is to follow one of these paths non-interactively, not knowing ahead of time which level the item is at (and thus which of the $\log_2 n$ paths will be followed).

To achieve this, we propose to have the Bloom filter results themselves be used in unlocking one of the two possible edges leading to the next query. A successful lookup will unlock the edge leading to a "finished" path, under which only fake queries will follow. Conversely, failure must unlock the edge continuing down the "active" search path. Once on the "finished" path, it is impossible to return back to the "active" path. Most importantly, the server must not gain any ability at identifying which path it is currently on.

One strawman idea, exponential in the number of Bloom filter hashes k, is to *make each bit in the Bloom filter a piece of a decryption key.* For each level, the client prepares 2^k results, corresponding to each possible state of the Bloom filter. The Bloom filter keys are generated deterministically by the client using a cryptographic hash, so that the client can efficiently keep track of them with only logarithmic storage. That is, a bit set to *1* at position *pos* in the Bloom filter is represented by $T_{pos} = \text{Hash}(sk \mid pos \mid \text{level} \mid \text{gen} \mid 1)$, and a bit set to *0* by $F_{pos} = \text{Hash}(sk \mid pos \mid \text{level} \mid \text{gen} \mid 0)$. The server learns only one of the two (*never both*).

A Bloom filter lookup involves k bit positions (k is the number of underlying Bloom filter hash functions). For each new level it traverses, the server needs to know the k associated Bloom filter bit positions to retrieve, constituting this level's query. *For the first level, these are provided by the client.* For each successive level, the server will get this information by incrementally decrypting portions of a client-provided "query object" data structure.

Illustrated in Figure 2 (right), the "query object" is composed of $\log n$ levels and is traversed by the server top-down synchronized with the traditional ORAM traversal. The query object allows the server to progress in its database traversal without learning anything.

Each level in the query object (with the exception of the root), contains two nodes: a "finished" node and an "active" node. Each node contains the k positions defining the current level Bloom filter query. The nodes also contain a "keying" set of 2^k elements.[4]

After performing the Bloom filter lookup, the server will be able to decrypt one of these elements (only). Once decrypted, this element contains a key to decrypt one of the query object's next level two nodes; it also contains the identifier for a current level item to return to the client.[5]

In effect this tells the server where to look next in the query object—i.e., which of the query object's next level two nodes ("finished" or "active") to proceed with. This guides the server obliviously down either the "finished" or the "active" path, as follows:

- If the current level contains the sought-after item, the server's work is in fact done. However, the server cannot be made aware of this. Hence, it is made to continue its traversal down the ORAM database, via a sequence of fake queries. The "finished" node of the next query object level allows the server to do just that, by providing the traversal information down the "active" path.
- If, however, the current level *does not* contain the sought-after item, the server must be enabled to further query "real" data in its traversal down the ORAM database—it will thus receive access to "active" node of the next query object level.

To prevent the server from decrypting more than one element from a node's "keying" set, a special encryption setup is deployed. Each of the 2^k elements of the "keying" set is encrypted with a special query object element key (QOEK),

[4] After encryption, these elements are sent in a random order to prevent the server from learning any information.

[5] To prevent leaks, the server will return one item for each level, since we do not want to reveal when and where we found the sought-after real item.

only one of which the server will be able to reconstruct correctly after its Bloom filter query.

More specifically, for a Bloom filter lookup resulting in k bit representations (i.e., bit_i is the representation of the bit at position i – either T_i or F_i [6]), the QOEK is defined as QOEK $= \text{Hash}(bit_1 \mid bit_2 \mid bit_3 \mid ... \mid bit_k)$.

The encryption setup of the "keying" set ensures that this key decrypts exactly one of its elements. The element corresponding to a Bloom filter "hit" (the sought-after element was found at this level, i.e., all the underlying Bloom filter bits are set to *1*) leads down the "finished" path, i.e., the element that QOEK decrypts now, leads down the "finished" path in the query object's next level.

5 Efficient Construction

We now present an efficient construction, with only $O(\log n)$ client storage, $O(\log n)$ per-query online message size, $O(\log^2 n \log\log n)$ amortized communication, and still only $O(1)$ round trips. We reduce the size of the query object of Section 4 from $2^k \log n$ to just $k \log n$.

The main insight is to allow compression of the 2^k decryption key possibilities into only $k+1$ possibilities. This is achieved by representing the Bloom filter bits and their combination in a commutative format. By allowing the decryption key pieces stored in the Bloom filter (described in the previous section) to be added together, rather than concatenated, the client only has to account for $k+1$ different outcomes at each level.

To this end, we start by first establishing a secret level-instance-specific token $v = \text{Hash}(sk \mid level \mid gen)$, not known to the server. In the Bloom filter, a bit set to *1* at position pos is represented as $T_{pos} = \text{Hash}(sk \mid level \mid gen \mid pos)$; a bit set to *0* is represented as $F_{pos} = T_{pos} + v \mod 2^{c_0}$ (Figure 3), where c_0 is a security parameter. In the following we will continue to operate in $\mathbb{Z}_{2^{c_0}}$. We also assume that $\text{Hash}(\cdot)$ operates in $\mathbb{Z}_{2^{c_0}}$.

Now, the query object encryption key (QOEK) is generated by combining (adding) values using modular arithmetic, instead of concatenation as in the strawman solution. This allows the client to only account for $k+1$ possibilities, each key corresponding to the number of times v might show up among the selected Bloom filter positions.

The server sums together the values found in the Bloom filter, and performs a hash, yielding, e.g., $\text{Hash}(bit_1 + bit_2 \ldots + bit_k \mod 2^{c_0})$ as the QOEK. The commutativity of modular addition means each permutation of bits set to *0* and bits set to *1* in a given Bloom filter yields the same key. A successful Bloom filter lookup occurs in the case that the QOEK is $\text{Hash}(T_{pos_0} + T_{pos_1} + ... + T_{pos_k} \mod 2^{c_0})$, which unlocks the edge from the "active" to the "finished" path.

Further, each of the k values from $\text{Hash}(T_{pos_0} + T_{pos_1} + ... + T_{pos_k} + v \mod 2^{c_0})$ through $\text{Hash}(T_{pos_0} + T_{pos_1} + ... + T_{pos_k} + kv \mod 2^{c_0})$—the result of a failed Bloom filter lookup (the server does not know they are failures)—unlocks an

[6] Recall that bit_i does not reveal to the server anything about the actual underlying Bloom filter bit.

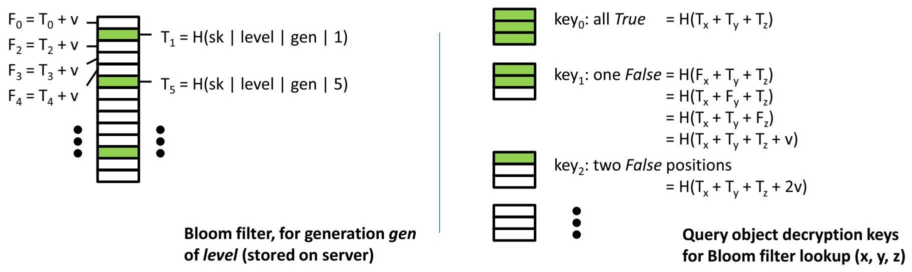

$F_0 = T_0 + v$
$F_2 = T_2 + v$
$F_3 = T_3 + v$
$F_4 = T_4 + v$
T_1 = H(sk | level | gen | 1)
T_5 = H(sk | level | gen | 5)
Bloom filter, for generation *gen* of *level* (stored on server)
key_0: all *True* = $H(T_x + T_y + T_z)$
key_1: one *False* = $H(F_x + T_y + T_z)$
= $H(T_x + F_y + T_z)$
= $H(T_x + T_y + F_z)$
= $H(T_x + T_y + T_z + v)$
key_2: two *False* positions
= $H(T_x + T_y + T_z + 2v)$
Query object decryption keys for Bloom filter lookup (x, y, z)

Fig. 3. Left: Bloom filter format. A bit set to *1* in the Bloom filter is represented by a hash of the position and current level key; a bit set to *0* is represented by the same value, plus the secret value v. Right: decryption keys used with the query object. The server obtains the decryption key for a given query stage by hashing together the specified Bloom filter results. Since there are k hash functions used in a Bloom filter check, the client includes in the query object an edge corresponding to each of the $k+1$ possible keys.

edge in the query object that results in continuing on the current ("finished" or "active") path (Figure 2). Figure 4 provides a summary of the query object format.

5.1 Obliviously building the Bloom filter and levels

This section describes how to construct the Bloom filter without revealing to the server which Bloom filter positions correspond to which items. Further, it shows how to obliviously scramble the items as a level is constructed. Both processes are mostly non-interactive (only a constant small number of round trips).

In the Bloom filter construction, the key privacy requirement is that the server is unable to learn ahead of time any correlation between Bloom filter positions and data items. Constructing a new level in a BF-based ORAM requires first randomly and obliviously permuting all the items, renaming them according to the new level hash function, and introducing a fake item for each (again, obliviously, and using the appropriate hash-function defined name). Constructing the Bloom filter requires first scanning the new set of items, building an encrypted list of positions that will be need to set, and then obliviously rearranging this encrypted list into the appropriately-sized segments of the resulting encrypted Bloom filter. The result is a pristine new level, generated using a deterministic read and write pattern (independent of the contents or history).

Further, note that we differ from previous work in that we store decryption keys in the Bloom filter, instead of single encrypted bits. Recall that these components are computed on the client by a secure keyed hash of the position, and added to the value v if this position is intended to represent a bit value of *0*.

The other main difference from existing work is that [14] assume a significant amount of temporary client storage, which is not suitable in our model. To avoid this requirement we propose to use two passes of an $O(n \log_2 n)$ oblivious randomized shell sort from [8].

- Level 1 active node:
  - Bloom filter lookup index positions $L1pos_1 \ldots L1pos_k$ (integer values)
  - the client computes, *but does not send*:
    - $key_{L1,\text{ success}} = \text{Hash}(T_{L1pos_1} + T_{L1pos_2} + \ldots + T_{L1pos_k} \mod 2^{c_0})$
    - $key_{L1,1} = \text{Hash}(T_{L1pos_1} + T_{L1pos_2} + \ldots + T_{L1pos_k} + v \mod 2^{c_0})$
    - $key_{L1,k} = \text{Hash}(T_{L1pos_1} + T_{L1pos_2} + \ldots + T_{L1pos_k} + kv \mod 2^{c_0})$
  - included in a random order:
    - $E_{key_{L1,\text{ success}}}$ (L1 active data ID, and key2F—key for L2 finished node)
    - $E_{key_{L1,1}}$ (L1 fake data ID and key2R—key for L2 active node)
    - $E_{key_{L1,k}}$ (L1 fake data ID and key2R—key for L2 active node)
- Both the L2 active and the L2 finished nodes, in a random order:
  - L2 real node, encrypted with key2R:
    - Bloom filter lookup index positions $L2pos_1 \ldots L2pos_k$
    - the client computes, *but does not send*:
      - $key_{\text{ success}} = \text{Hash}(T_{L2pos_1} + T_{L2pos_2} + \ldots + T_{L2pos_k} \mod 2^{c_0})$
      - $key_1 = \text{Hash}(T_{L2pos_1} + T_{L2pos_2} + \ldots + T_{L2pos_k} + v \mod 2^{c_0})$
      - $key_k = \text{Hash}(T_{L2pos_1} + T_{L2pos_2} + \ldots + T_{L2pos_k} + kv \mod 2^{c_0})$
    - included in a random order:
      - $E_{\text{ success}}$ (L2 real data ID, and key for L3 finished node)
      - E_{key_1} (L2 finished data ID and key for L3 active node)
      - E_{key_k} (L2 finished data ID and key for L3 active node)
  - L2 finished node, encrypted with key2F:
    - k random Bloom filter lookup index positions $L2pos_1 \ldots L2pos_k$
    - the client computes, *but does not send*:
      - $key_0 = \text{Hash}(T_{L2pos_1} + T_{L2pos_2} + \ldots + T_{L2pos_k} \mod 2^{c_0})$
      - $key_1 = \text{Hash}(T_{L2pos_1} + T_{L2pos_2} + \ldots + T_{L2pos_k} + v \mod 2^{c_0})$
      - $key_k = \text{Hash}(T_{L2pos_1} + T_{L2pos_2} + \ldots + T_{L2pos_k} + kv \mod 2^{c_0})$
    - included in a random order:
      - E_{key_0} (L2 fake data ID and key for L3 random node)
      - E_{key_1} (L2 fake data ID and key for L3 random node)
      - E_{key_k} (L2 fake data ID and key for L3 random node)
- Both the L3 active and the L3 finished nodes, in a random order etc.
- And so forth.

Fig. 4. Query Object Format. For each level, the query object is composed of two possible nodes (with the exception of the root/top which only has one node), and is constructed thusly. This set constitutes a total of $2 \log n$ nodes (containing associated Bloom filter requests), and $2k \log n$ edges. Of these nodes, the server will be able to unlock $\log n$. Each of the unlocked ones provides k edges, of which the server will be able to unlock exactly one. This contains the decryption key for a single node at the next level, as well as the data item ID to retrieve etc.

This shell sort will be applied to a list produced by the client as follows. The client starts by producing a list of (encrypted) positions that need to be set in the Bloom filter. The client will then also add a number of "segment delimiters" to this list. These delimiters will aid later. One delimiter is issued per segment (e.g., 32 adjacent positions) in the Bloom filter. These delimiters will later provide an excuse to output something for positions that are *not* to be set, to prevent the server from learning which bits are set.

The client then performs a first sorting pass, outputting the list sorted by positions, with the delimiters interspersed (delimiters include information that allows their sorting). This sorted list is then scanned, and, for each of its non-delimiter elements, a fake 32 bit value is issued. For each encountered segment delimiter however, a 32 bit (encrypted) segment of the Bloom filter is output.

This segment's bits are set correctly according to the recently seen (since the last segment's delimiter encounter) encrypted set positions.

This simple mechanism prevents the server from learning how many bits are set in each segment. To complete the process, a second oblivious sort pass then moves the fake 32 bit values to the end of this new list, where they can be safely removed by the server.

Finally, the encrypted bit-storing Bloom filter needs to be converted into a key-storing Bloom filter in one final step: in a single non-oblivious pass, we read each bit and output either T_{pos} for *1*s and F_{pos} for *0*s (where *pos* is the current bit's position in the Bloom filter). Note this multiplies the size of the remotely stored object by the key size.

We do not need to modify the level construction from [14], except in replacing their storage-accelerated merge sort with the storage-free randomized shell sort.
Non-interactivity of sort. It is worth noting that the shell sort, like the storage-accelerated merge sort, can be implemented in only a handful of round trips. Each step in both sorting algorithms requires reading two items from the server, and writing them back, possibly swapped (which is an interactive process). However, the item request pattern (of both sorting algorithms) is known ahead of time to the server, so it can send data to the client ahead of time, without waiting for the request from the client (likewise, the client can issue requests for future items far in advance of the time they will be used). This process is trivial in the merge sort: the access pattern consists of simultaneous scans of two (or sometimes up to $\sqrt{n}$) arrays; the server streams these to the client. This process of non-interactive streaming of sort data to the client is not as trivial in the randomized shell sort. Once the random seed is chosen, however, both the client and server know in advance the order the items will be requested in. The server can run a simulation of this sort, for example, to know which items the client needs to read next, and avoid waiting for network round-trips throughout the construction of the level.

6 Security

SR-ORAM directly inherits the privacy properties of the server-side ORAM database traversal, as well as the integrity defenses from the base ORAM construction in [14]. We must now establish the privacy of the query object construction, as well as the new Bloom filter construction.

We establish privacy of the query object construction in Theorem 1. The server learns only one set of Bloom filter positions and one item label to retrieve at each level for a given query. In other words, the server sees only what it would see in an equivalent, interactive instantiation.

Lemma 1. *The server gains no non-negligible advantage at guessing $v = Hash\ (sk \mid level \mid gen)$ from observing (i) the Bloom filter contents or (ii) the hashes included in the query object.*

Lemma 2. *For a constant $u > 1$, a Bloom filter using k hashes, with the size-to-contained items ratio of $k \times u$, has a false positive rate bounded by u^{-k}.*

Lemma 3. *For a security parameter c_1, a constant $u > 1$, and a number of $\log n$ lookups on Bloom filters with size-to-contained items ratios of $k \times u$, where $k = c_1 + \log_u \log n$ hashes, the overall false positive rate is negligible in c_1.*

Theorem 1 *The server can only unlock one path down the query object, and all paths appear identical to the server.*

Theorem 2 *The server learns nothing from the Bloom filter construction.*

Theorem 3 *An honest but curious adversary gains no non-negligible advantage at guessing the client access pattern by observing the sequence of requests.*

Theorem 4 *An actively malicious adversary has no advantage over the honest but curious adversary at violating query privacy.*

7 Analysis

Following from the construction in Section 5, the query object size is $O(\log n)$. This is transmitted to the server, which performs $k \log n$ decryption attempts (of which $\log n$ are successful) before sending $\log n$ blocks back to the client. This yields the online cost of $O(\log n)$.

The amortized offline cost per query considers the time required to build each level. A level sized z is built twice every z queries. Shuffling these items using a randomized shell sort costs $O(z \log z)$. Since, as shown in Lemma 3, the Bloom filter is sized $z \log \log n$, and it must also be shuffled, the Bloom filter construction cost of $O(z \log z \log \log n)$ dominates asymptotically. Summing over the i levels sized $z = 4^i$, and amortizing over the queries for each level between rebuilding, we find a total amortized offline cost of $O(\log^2 n \log \log n)$

A query requires a single online round trip: the client generates a query object, sends it to the server, and receives a response containing the answer. The offline shuffle process requires several round trips (as discussed in Section 5.1), but this cost is amortized over a period corresponding to many queries, so that the average number of round trips per query is still well under 2.

8 Acknowledgements

We would like to thank our anonymous reviewers, who provided helpful comments about the manuscript.

9 Conclusion

We introduced a new single-round-trip ORAM. We analyzed its security guarantees and demonstrated its utility. While non-interactivity is of significant theoretic interest in itself, we also showed this to be the most efficient storage-free-client ORAM to date for today's Internet-scale network latencies.

References

1. M. Ajtai, J. Komlos, and E. Szemeredi. An O(n log n) sorting network. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 1–9, 1983.
2. Dimitri Asonov. *Querying Databases Privately: A New Approach to Private Information Retrieval*. Springer Verlag, 2004.
3. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
4. Dan Boneh, David Maziéres, and Raluca Ada Popa. Remote oblivious storage: Making Oblivious RAM practical. Technical report, MIT, 2011. MIT-CSAIL-TR-2011-018 March 30, 2011.
5. W. Gasarch. A WebPage on Private Information Retrieval. Online at `http://www.cs.umd.edu/~gasarch/pir/`.
6. Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on Oblivious RAMs. *Journal of the ACM*, 45:431–473, May 1996.
7. Michael Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Oblivious RAM simulation with efficient worst-case access overhead. In *ACM Cloud Computing Security Workshop at CCS (CCSW)*, 2011.
8. Michael T. Goodrich. Randomized shellsort: A simple oblivious sorting algorithm. In *Proceedings 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.
9. IBM. IBM 4764 PCI-X Cryptographic Coprocessor (PCIXCC). Online at `http://www-03.ibm.com/security/cryptocards/pcixcc/overview.shtml`, 2006.
10. Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based Oblivious RAM and a new balancing scheme. Cryptology ePrint Archive, Report 2011/327, 2011.
11. Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In *CRYPTO*, pages 502–519, 2010.
12. Emil Stefanov, Elaine Shi, and Dawn Song. Towards Practical Oblivious RAM. In *To Appear in Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2012.
13. Shuhong Wang, Xuhua Ding, Robert H. Deng, and Feng Bao. Private information retrieval using trusted hardware. In *Proceedings of the European Symposium on Research in Computer Security ESORICS*, pages 49–64, 2006.
14. Peter Williams, Radu Sion, and Bogdan Carbunar. Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In *ACM Conference on Computer and Communications Security*, pages 139–148, 2008.

Extended Abstract: Cipher Techniques to Protect Anonymized Mobility Traces from Privacy Attacks

Chris Y. T. Ma[♯], David K. Y. Yau[◇♯], Nung Kwan Yip[◇], and Nageswara Rao[‡]

[♯] Advanced Digital Sciences Center, Illinois at Singapore
[◇] Purdue University, West Lafayette, IN, USA
[‡] Oak Ridge National Laboratory, TN, USA

Abstract. We study the problem of privacy protection of openly published mobility traces of people and other real-world entities, beyond standard protection techniques of anonymization and spatial/temporal cloaking. It has been shown that an adversary, armed with side information learned from real-world channels, could use powerful Bayesian inference methods to compromise the standard protection significantly [1]. In this paper, we argue that a major challenge of more effective protection is the need to balance between *secrecy* for protection and *transparency* for functional usefulness of the protected traces. We present "classical cipher" techniques applied to the spatial and/or temporal domains to give a useful tradeoff between the two antagonistic requirements. Driven by real-world mobility traces, diverse simulation results illustrate the performance of the proposed *location cipher* and *time-zero cipher*. We show that these ciphers can mitigate the privacy problem, and the ciphers work best when combined.

1 Introduction

Mobility traces of people, vehicles, and other real-world entities are valuable for numerous purposes. In marketing, knowing the places likely to be visited by individuals in sequence would allow marketers to plan their advertising strategies and custom design messages sent to say users' smartphones to sell products or services. In the research community, the usefulness of mobility traces has led to their widespread publication for general use by researchers and system designers.

The public availability of information about individuals has led to natural privacy concerns, even though the true identities of participants in traces are made anonymous. The concerns arise because mobility traces do not exist in a vacuum. Rather, they give information about individuals who can be openly observed in public places and who engage in diverse communication channels – everyday conservations, written articles, twitters, and blogs – to reveal snapshots of their whereabouts. Armed with such "side information" about the locations of participants (of known identities) in a mobility trace, a curious individual, whom we call an "adversary," could infer the extended location information of one or more of these participants.

In view of the privacy problem, publishers of mobility traces have resorted to standard "cloaking" techniques which introduce random noise in the spatial and/or temporal dimensions to reduce the precision of recorded data points and hence their linkability to real-world side information, which is itself oftentimes noisy due to imperfect references and/or learning. It turns out, however, that common noise models are not quite sufficient to confound the adversary [1]. This is because, faced with the noise, an intelligent adversary could employ powerful Bayesian inference techniques to arrive at high confidence conclusions nevertheless, much like robust sensing and statistical estimation under uncertainty.

For strong protection of secrets (in our case, the true identities of anonymous participants in a trace), standard cryptography based on one-way functions could be used [2], [3]. For mobility traces, however, privacy must be protected against attacks by even a legitimate user of the traces. Hence, on the one hand, the user may not be given a key to unlock the secret. On the other hand, the unavailability of the key implies that the traces in "encrypted" form must preserve sufficient original information to remain useful to the user. The last requirement rules out strong cryptography as a protection strategy.

In this paper, we tackle the problem of balancing between *secrecy* for privacy protection and *transparency* for functional effectiveness by introducing a prudently chosen puzzle (or a "cipher") to encrypt (or "transform") the trace beyond the introduction of commonly applied statistical noise. The chosen cipher serves one key purpose based on important features of the mobility problem domain – it aims to reduce the linkability between recorded trace data points and any side information obtained by the adversary.

We present a set of specific proposed ciphers that realizes a spectrum of tradeoff between the strength of privacy protection and the range of applicability of the transformed traces in different concrete problems. We classify a cipher according to whether or not it randomizes (A) the identities of recorded locations (if so, we call the cipher a *location cipher*) and (B) the absolute time of recorded sampling points (if so, we call the cipher a *time-zero cipher*). Notice that the location cipher A can be used either by itself, or in combination with cipher B.

Our **contributions** are as follows. (i) We propose a set of protection ciphers for the privacy of mobility traces. Importantly, these ciphers preserve information needed by various real-world applications *by design*, and hence can increase privacy without compromising usefulness. (ii) We design adversary strategies that aim to solve the ciphers in conjunction with Bayesian techniques to infer a victim's trace under noise, while considering all available information in diverse forms. (iii) Our quantitative results show that applied individually, the ciphers A and B can clearly mitigate the privacy attack.

2 Problem Definition

We assume that a set of traces, each recording intermittently the time and corresponding location of a mobile node, are published to the public. We call a mobile node that is included in the published trace set a *participant* in the trace set. We assume that each trace entry is given in the format of $\{id, time, location\}$, where id is a random and unique ID replacing consistently the true identity of the mobile node, $time$ is the sampling time of the node's location, and $location$ is that recorded location. We assume that *spatial cloaking* of the location information has been applied to the traces before publication, for increased privacy protection. Specifically, the traced area is divided into a grid of cells, each of size $S \times S$, and *location* is published as the cell ID instead of a more precise point within the cell. Further to the spatial cloaking, for privacy protection, we study two cipher techniques that may be applied to transform the original traces before publication. Details of these techniques are presented in Section 4.

In addition, we assume that there is an *adversary* whose purpose is to learn the true identities of one or more participants in the trace set. To assist this endeavor, the adversary has learned *snapshots*, in the form of $\{time, location\}$ pairs (where *time* and *location* are possibly noisy real-world information), of the whereabouts of each potential *victim* who is a trace participant. In addition, the adversary is equipped with background (e.g., common sense) knowledge about the world, such as the geography of the traced area, the popularity of different locations in this area, and the general mobility patterns and preferences of the participants.

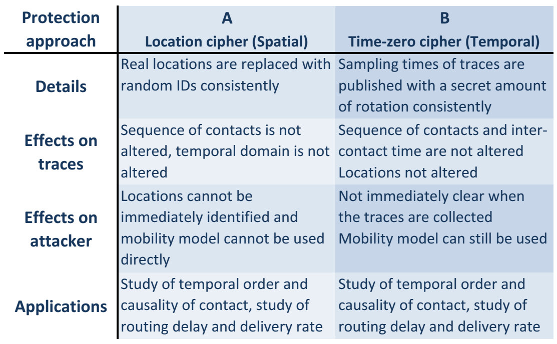
3 Related Work

Data privacy is well studied in the database literature [4], [5]. Sweeney [4] proposes a protection model named *k-anonymity*, as well as a set of accompanying policies for the privacy protection. When k-anonymity is satisfied, each individual is indistinguishable from $k - 1$ other individuals. Xiao and Tao [5] propose a generalization principle of *m-invariance* to effectively limit the risk of privacy disclosure in data re-publications, given the many potential correlations among various snapshots of each data entry in subsequent publications that could be used to derive sensitive information.

Privacy protection of mobile nodes in location-based services has also received attention [6], [7], [8]. One proposed approach is to reduce the spatial/temporal granularity of the location information made available to the service provider, while achieving satisfactory service effectiveness [6]. Hoh *et al.* [7] devise a protection strategy to release user data only when certain privacy constraints are met. Meyerowitz and Choudhury [8] suggest sending fake requests with the real ones to reduce the ability of an eavesdropper to trace a mobile node over time.

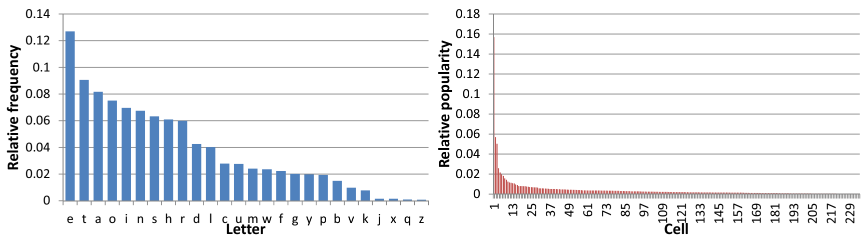
Various approaches have also been published in the literature to address the privacy concern in publishing geo-located datasets. They include data perturbation, and data generalization or granularity reduction. Nergiz *et al.* [9] extend the notion of k-anonymity to trajectories, and propose a generalization-based approach to publish trajectories for enhanced privacy protection. Abul *et al.* [10] propose the use of space translation to achieve (k, δ)-anonymity for moving object databases, where δ is the radius of a cylindrical volume representing the trajectory imprecision.

The solutions proposed in these papers do not suit our purposes because (1) some of them publish only summary statistical information about the traces [9], (2) some reduce the precision of the trace information solely for the purpose of making traces indistinguishable from each other but without regard for the application requirements [6], [7], [10], and (3) some do not preserve sufficient fidelity of information about the movements of individual nodes (e.g., they may reorder events, or introduce bogus events) [8]. In particular, nodal contact information



Protection approach	A Location cipher (Spatial)	B Time-zero cipher (Temporal)
Details	Real locations are replaced with random IDs consistently	Sampling times of traces are published with a secret amount of rotation consistently
Effects on traces	Sequence of contacts is not altered, temporal domain is not altered	Sequence of contacts and inter-contact time are not altered Locations not altered
Effects on attacker	Locations cannot be immediately identified and mobility model cannot be used directly	Not immediately clear when the traces are collected Mobility model can still be used
Applications	Study of temporal order and causality of contact, study of routing delay and delivery rate	Study of temporal order and causality of contact, study of routing delay and delivery rate

Table 1. Overview summary of and comparison between the proposed protection ciphers.



(a) English text (b) 235 San Francisco cab locations

Fig. 1. Comparison of symbol frequency between popularity of San Francisco locations for cabs and frequency of letters in general English text.

is lost in the published traces, rendering them unsuitable for the applications discussed in Sections 1 and 4.

4 Protection Measures

Apart from anonymizing the true identities of participants in mobility traces, transformation techniques can be applied to other attributes of the traces. In this paper, we focus on classical cipher solutions. Specifically, we will study the location cipher and the time-zero cipher mentioned in Section 1. Definitions of these ciphers will be presented below, but an overview summary of their properties is given in Table 1.

4.1 Location cipher (A)

In this technique, each real-world location in the original trace is replaced consistently by a unique and random ID in the published traces. The random IDs are not correlated in any way with the real locations. The IDs can be generated by using a hash or cipher function on the locations' geographical coordinates.

With encrypted locations, the adversary cannot attack by directly comparing the side information snapshots with the published traces, and mobility inference is not possible since the geography of the locations is lost. To overcome the problem, the adversary would need to solve the location cipher first, before further attack actions. Substitution ciphers have been used to encrypt English text, where they could be attacked by say frequency analysis of different letters in the alphabet. Similar attacks could be launched in our problem domain, but they face significantly increased challenges. First, there are far more possible locations than letters in the English alphabet. Second, it is less likely for an adversary to have fine-grained distinction between cells based on popularity. Indeed, as shown in Fig. 1 for cabs in San Francisco, many real-world locations would have very similar popularity. Unlike letters in the English alphabet, with the exception of a few very popular ones, most locations are almost indistinguishable from each other based on frequency analysis. Hence, we expect that our adversary will have significantly lower confidence in solving the location cipher.

4.2 Time-zero cipher (B)

In this technique, the sampling times of locations in a trace are not published in absolute values, but they are all relative to a start time, t_0, whose true value is not released. For example, if a location sample is taken at 4pm and the unknown start time is 2pm on the same day, then the published sample time will be the $(t_0 + 2)$-th hour instead of 4pm.

Time-zero cipher is solved if the absolute time of any of the recorded samplings is revealed, so that the adversary can determine the time shift applied. Although the adversary cannot learn absolute time directly from the traces, she may still use *sub-sequence matching* [11] to infer the trace corresponding to a victim, especially if she has collected a long enough sequence of the side information snapshots. In this method, the adversary treats the collected snapshots as a "sub-sequence" of symbols and looks for its occurrence in each trace treated as a complete sequence.

5 Adversary Strategy

In this section, we detail the adversary's attack strategy. We assume that the trace set has been protected by (i) the location cipher only, (ii) the time-zero cipher only, or (iii) both the location and time-zero ciphers. The adversary knows which of the above three situations she is facing.

5.1 Attacking the location cipher

To solve the location cipher, we assume that the adversary has general world knowledge about the relative popularity of locations in the traced area (corresponding to an order-0 Markov model of the mobility) and the patterns of movements between adjacent locations (order-n Markov model), which captures the physical constraints of the movements as well as preferences of nodes in moving between locations.

In the attack, the adversary first computes, from the published trace set, the statistics of location popularity and the transition probabilities from one location to the next. After that, she compares the computed statistics from the traces with her general knowledge of the real-world statistics. The attack consists of two steps and works as follows:

Frequency analysis of symbols (FA-0). According to her general world knowledge captured in the order-0 Markov model, the adversary ranks the real-world locations in decreasing order of popularity. According to what she computes from the trace sets, the adversary ranks the randomized locations in decreasing order of popularity. For each randomized location, the adversary assigns the correspondingly ranked real-world location as a *tentative solution* for that randomized location. Notice that we cannot have more randomized locations than real-world locations in the order-0 Markov model. Hence, every randomized location must be assigned a (possibly tentative) solution.

Frequency analysis of sequences up to order-n (FA-n). Sometimes, the tentative solutions assigned in the previous step are not reliable as the successive randomized locations (rank-ordered by in the above step) differ little by their popularity. In this case, we put randomized locations with similar popularity into an equivalence set $\mathcal{L}$, and their tentative solutions into a corresponding equivalence set $\mathcal{C}$ for further evaluations. For each randomized location $i \in \mathcal{L}$, we call $\mathcal{C}$ its *candidate solution set*, and each solution $j \in \mathcal{C}$ is assigned a probability, q^i_j, representing the likelihood for j to be the solution of i. We define $\mathcal{Q}^i \triangleq \{(j, q^i_j) : j \in \mathcal{C}\}$ as the *likelihood distribution* for the solutions of i. To refine the solutions for locations in the same equivalence set, we use higher-order (up to order-n) Markov knowledge of the real-world movement and the corresponding computed statistics from the trace set.

Notice that increasingly higher-order Markov knowledge could be used recursively to refine the solutions in principle. In practice, however, such detailed information may not be beneficial to the adversary, as it is often noisy and represents mainly the average information about all the mobile nodes instead of individual ones.

5.2 Identifying the victim's trace using Bayesian inference

In this part of the attack, the goal of the adversary is to identify in some optimal fashion the complete path history of a victim, given some snapshots of the victim. We divide the discussion into cases according to whether the times of the snapshots are known to coincide with the sampling times in the traces, and whether or not the traces have been protected with the location cipher and/or time-zero cipher.

Without any cipher This is the case studied in [1] in which Bayesian inference is used to overcome the noise in the spatial domain of the snapshots and/or the traces, and return a robust answer of the victim's trace to the adversary. In particular, it is shown in [1] that the **maximum likelihood estimator (MLE)** method is a robust and effective attack strategy, when information is noisy but the adversary could have some estimation about the model of the noise perturbing the traces and/or snapshots.

Considerations for the location cipher (A) When the location cipher is used, the structure and the equations of the Bayesian inference are kept. However, as mentioned before, each randomized location in a published trace is in

	Cells visited	# nodes	Time
SF cabs	3997	536	17/05/2008 - 10/06/2008

Table 2. Parameters of the real traces.

general resolved by the cipher only up to a likelihood distribution (Section 5.1). To account for this uncertainty, we generalize the compatibility value between a snapshot and a trace to be the expectation of compatibility values over candidate solutions of relevant randomized locations recorded in the trace.

Note that in the attack, the Bayesian inferencing makes use of *all* available snapshots, whether they are collected at popular locations to which relevant encrypted locations can be resolved with high confidence, or at unpopular locations to which the encrypted locations can only be resolved to uncertain likelihood distributions.

Considerations for time-zero cipher When the time-zero cipher is used, the adversary uses sub-sequence matching [11] as a basis of her attack strategy.

6 Performance Evaluation

In this section, we evaluate the effectiveness of the protection ciphers by quantifying the effectiveness of the privacy attack by the adversary. The results reported are averages for simulation experiments each repeated 1,000 times using the San Francisco cab traces [12]. Parameters of these traces are listed in Table 2.

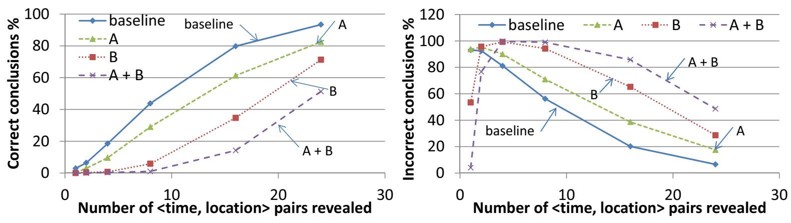
We study the performance of the privacy attack in Section 5.2. If the traces are protected with the location cipher, we assume that the adversary has already deciphered the random location IDs, even though the cipher attack may not be complete or correct. We assume that the adversary then uses the **maximum likelihood estimator (MLE)** inference strategy. We study the special case where the snapshots of the victim collected by the adversary coincide with the sampling times of the traces.

We quantify the performance of the strategies with the following metrics, (i) *Correct conclusion percentage.* A conclusion is correct if the victim is uniquely identified by the adversary according to the criterion of highest compatibility metric; (ii) *Incorrect conclusion percentage.* A conclusion is incorrect when the victim is not among the set of candidates having the highest compatibility metric.

In the simulations, we assume that the spatial granularity is of 0.01° in latitude and longitude, the snapshots are perturbed with zero-mean Gaussian noise with $\sigma = 5$, and a randomized/real-world location is unpopular if its popularity is less than 0.01% of the total length of visit duration.

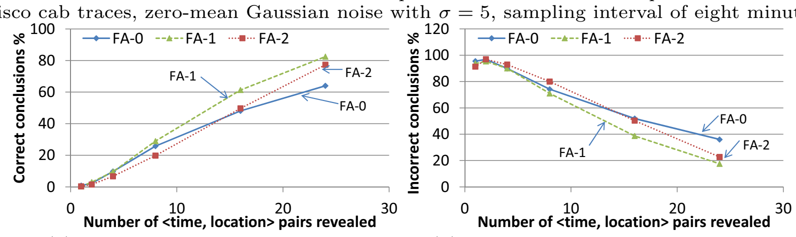
6.1 Comparison of attack performance between various ciphers (A, B, A + B)

Fig. 2 compares the performance of the attack strategy used by the adversary when the anonymized traces use different approaches for privacy protection, namely the location cipher and/or the time-zero cipher for the cab traces. We assume that the adversary only has order-0 and order-1 Markov model knowledge about the movement of the mobile nodes. The curve with the label *baseline* in



(a) Fraction of correct conclusions (b) Fraction of incorrect conclusions

Fig. 2. Performance of attack when traces are protected with different ciphers. $S = 1$ km, San Francisco cab traces, zero-mean Gaussian noise with $\sigma = 5$, sampling interval of eight minutes.



(a) Fraction of correct conclusions (b) Fraction of incorrect conclusions

Fig. 3. Performance of attack when the adversary has different details of world knowledge about the mobility. Traces are protected with the location cipher. $S = 1$ km, San Francisco cab traces, zero-mean Gaussian noise with $\sigma = 5$, sampling intervals of eight minutes.

the two figures corresponds to the performance of the privacy attack studied in [1] when the traces are not protected with any ciphers.

Summary of main results. (i) The location cipher A clearly mitigates the privacy attack in all the experiments (Fig. 2), compared with the baseline of no cipher protection. (ii) In general, the time-zero cipher B is more effective than A. This is observed in Fig. 2 for as many as 16 side-information snapshots. However, since the cipher B attack is based on subsequence matching, it can benefit a lot from a longer sequence of snapshots. (iii) The combined A+B cipher performs the best: The attacker fails to identify the victim at all with up to 8 pieces of side information, and her accuracy drops from 80% to 15% for the cab traces (Fig. 2(a)). This shows that the A+B cipher offers highly effective protection in practical attack scenarios.

6.2 Comparison of attack performance with different order of general world knowledge

Given that the traces are location-cipher protected, we further evaluate the performance of the attack strategy when the adversary has different details of knowledge about the topology of the traced area and the mobility preferences of the nodes.

Fig. 3 shows the results when the adversary has different details of knowledge. In the figure, *FA-0* means that the adversary only knows the popularity of the locations but not the order-1 mobility model; *FA-1* means that the adversary also knows the order-1 Markov model; and *FA-2* means that the adversary also knows the order-2 Markov model. We can observe from the figure that as the adversary gains order-1 Markov model knowledge, the attack becomes more effective. However, using additional knowledge from the order-2 Markov model, the effectiveness of the attack is *reduced* (Fig. 3).

7 Conclusion

We have presented a set of ciphers to protect the privacy of mobility traces while retaining their usefulness for different applications. These ciphers can be applied to the spatial dimension, temporal dimension, or both of them at the same time. We have assumed an intelligent adversary, and studied how she can exploit information in the traces, in general world knowledge, and in real-world snapshots obtained about potential victims, to increase the effectiveness of her attacks. We provide simulation results, driven by real-world mobility traces, to demonstrate that the ciphers, applied individually or in combination, may significantly increase the effectiveness of privacy protection, compared with, in particular, the results in [1], where privacy protection is by standard cloaking of spatial and temporal information in the original traces.

References

1. Ma, C.Y.T., Yau, D.K.Y., Yip, N.K., Rao, N.S.V.: Privacy Vulnerabilities of Published Anonymous Mobility Traces. In: ACM MobiCom, Chicago, IL (September 2010)
2. : Data Encryption Standard. Federal Information Processing Standards Publication 46-1 (1977)
3. Jonsson, J., Kaliski, B.: Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447, Internet Engineering Task Force (February 2003)
4. Sweeney, L.: k-anonymity: a Model for Protecting Privacy. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems **10**(5) (2002)
5. Xiao, X., Tao, Y.: M-invariance: Towards Privacy Preserving Re-publication of Dynamic Datasets. In: ACM SIGMOD, Beijing, China (June 2007)
6. Gruteser, M., Grunwald, D.: Anonymous Usage of Location-Based Services through Spatial and Temporal Cloaking. In: ACM MobiSys, San Francisco, CA (May 2003)
7. Hoh, B., Gruteser, M., Xiong, H., Alrabady, A.: Preserving Privacy in GPS Traces via Uncertainty-Aware Path Cloaking. In: ACM CCS, Alexandria, VA (October 2007)
8. Meyerowitz, J., Choudhury, R.R.: Hiding Stars with Fireworks: Location Privacy through Camouflage. In: ACM MobiCom, Beijing, China (September 2009)
9. Nergiz, M.E., Atzori, M., Saygin, Y., Guc, B.: Towards Trajectory Anonymization: a Generalization-Based Approach. Transactions on Data Privacy **2**(1) (2009)
10. Abul, O., Bonchi, F., Nanni, M.: Never Walk Alone: Uncertainty for Anonymity in Moving Objects Database. In: IEEE ICDE, Cancun, Mexico (April 2008)
11. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast Subsequence Matching in Time-Series Databases. In: ACM SIGMOD, Minneapolis, MN (May 1994)
12. Piorkowski, M., Sarafijanovic-Djukic, N., Grossglauser, M.: CRAWDAD data set epfl/mobility (v. 2009-02-24). Downloaded from http://crawdad.cs.dartmouth.edu/epfl/mobility (February 2009)

An Ad Hoc Group Signature Scheme for Accountable and Anonymous Access to Outsourced Data

Wensheng Zhang and Chuang Wang

Department of Computer Science
Iowa state University, Ames, IA, USA
Email:{wzhang, chuangw}@iastate.edu

Abstract. This paper presents a group signature scheme called AdHocSign for dynamically formed groups, to support accountable and anonymous access to outsourced data. Each user is assigned a certain set of attributes and the secrets associated with these attributes when joining the system; the access to outsourced data is regulated by a logical expression of attribute which defines an ad hoc group of authorized users. The proposed AdHocSign enables a user who is allowed to access a piece of data to authenticate himself/herself to the host of the data using its preloaded secrets and some auxiliary information provided by the host; no extra secrets are needed to be distributed to the users when new access structures are constructed or existing access structures are modified. The selfless-anonymity and traceability of the AdHocSign scheme has been proved based on the hardness assumption of the q-SDH and the Decisional Linear problems.

1 Introduction

With wide application of Internet, especially the increasing adoption of the cloud computing paradigm, storing sensitive user data to un-trusted, remote hosts on Internet has been popular. The outsourced sensitive data are often stored in an encrypted form to protect the confidentiality from their hosts. To realize fine-grained control of accesses to the encrypted data, attribute-based encryption schemes (ABE) [1,8,3] have been proposed. Particularly, with the ciphertext-policy ABE scheme (CP-ABE) [3], each user owns a set of attributes and a set of secrets derived from the attributes. When a piece of encrypted data is outsourced to a host, the data is labeled with an access structure to specify who are allowed to access the data; the access structure is expressed in attributes and logical operations over the attributes. A user is allowed to access certain data if and only if the attributes owned by the user satisfy the access structure of the data.

In addition to fine-grained access control, the ABE schemes also supports anonymity as well as system dynamism and scalability. Specifically, a user can access the data that he/she is authorized to access without exposing his/her identity to the data host; also, when a new access structure is defined for certain encrypted data, there is no need to distribute new attributes or secrets to users or change the attributes or secrets owned already by users. The ABE schemes, however, do not provide accountability, which is necessary to trace out misbehaving users and stop them from abusing the privacy preservation features.

To provide user accountability while supporting access control and user anonymity, the existing group signature schemes [2,5,7] may be applied in some simple scenarios.

Particularly, if an access structure is defined as only one attribute, i.e., who are allowed to access a piece of data can always be specified as the set of users who own a certain attribute, the group signature schemes can be applied as follows: For each attribute, all users owning the attribute form a group. A group public key can be computed for the group by a trusted authority. When a user is given this attribute, the trusted authority computes a unique private key and gives it to the user. For a host hosting some data that can only be accessed by users owning this attribute, it is given the public key of the group. Based on the above key distribution, each user owning this attribute can authenticate himself/herself to the data host without exposing his/her identity. When needed, the authority is able to reveal the identify of authenticated user according to the messages (i.e., signature) sent by the user during the authentication process.

If the access structure of the data is complex, however, applying the group signature schemes may be inefficient or infeasible. Suppose the access structure for certain data is $a_1 \wedge a_2 \wedge (a_3 \vee a_4)$, meaning only users who own attribute a_1, a_2 and either a_3 or a_4, are allowed to access the data. To apply the group signature scheme, an *ad hoc group* may have to be formed to include all the users who are allowed to access the data according to the above access structure.

We propose a new signature scheme, named *AdHocSign*, for ad hoc groups that are defined dynamically according to access structures. The key ideas are as follows: (i) Instead of distributing group private keys to the members of an ad hoc group when the group is created (i.e., when a new access structure is defined), the new scheme pre-loads some key materials to individual users when they join the system and are given attributes. (ii) When certain data is posted to a host, the host is given certain auxiliary information that is computed by a trusted authority according to the access structure of the data. (iii) When a user needs to access a piece of data, it contacts the host of the data to obtains the access structure of the data and the afore-mentioned auxiliary information pre-loaded to the host by the authority. If the user's attributes satisfy the access structure (i.e., the user is a member of the ad hoc group defined by the access structure), the user can compute his/her own private key for the ad hoc group based on the auxiliary information and pre-loaded key materials, and authenticates himself/herself to the host. The user does not expose his/her *identity* or *ownership of attributes* (unless the ad hoc group is defined based on a single attribute or a conjunction of attributes) during the authentication.

In the following, background and formal overview of the AdHocSign scheme are presented in Section 2. Sections 3 and 4 elaborate the design and analysis of the AdHoc-Sign scheme for conjunction-only access structures and for general access structures. Section 5 summarizes related work and Section 6 concludes the paper.

2 Preliminaries

System Model We consider a distributed system composed of one or multiple data storage sites (called data *hosts*), multiple users, and an authority trusted by all the users and hosts. To facilitate access control to the data stored at the hosts, a set of attributes are defined. Each user e has a unique identity number denoted as $x_e \in \mathbb{Z}_p$, where $\mathbb{Z}_p$ is a finite field of p elements and p is a large prime number, and is assigned a set of attributes denoted as $\{a_{e,1}, \cdots, a_{e,n_e}\}$.

When a piece of data is posted to a host, who are allowed to access the data is specified as a logical expression containing attributes and conjunction ($\wedge$) or disjunction ($\vee$) operators on the attributes. We call the logical expression an *access structure*. Generally, an access structure T can be defined as

$$T = DT_1 \wedge \cdots \wedge DT_s, \tag{1}$$

where

$$DT_i = a_{T,i,1} \vee \cdots \vee a_{T,i,s_i} \tag{2}$$

for each $i = 1, \cdots, s$. For example, $T = a_1 \wedge (a_2 \vee a_3) \wedge (a_4 \vee a_5 \vee a_6)$ is an access structure defined based on attributes $a_1, \cdots, a_6$.

As an access structure can be defined on the fly when a piece of data is posted, the group of users allowed to access the data is not predefined; hence, we call such a group an *ad hoc group* and our proposed scheme is to provide a mechanism for members of such a group to sign messages in an anonymous and accountable manner. To protect the confidentiality of the data, we assume a certain encryption scheme, for example, the CP-ABE scheme [3], is used to encrypt the data to prevent the host and unauthorized users from decrypting the data.

To summarize, when a user in our system wants to access a piece of data, he/she first authenticates himself/herself to the host of the data using our proposed ad hoc group signature (*AdHocSign*) scheme. After the authentication succeeds, the data, in the encrypted form, is returned to the user, who can decrypt the data using the ABE schemes. To facilitate the authentication and data decryption, a user is assigned some attributes and secrets associated with the attributes when he/she joins the system.

Bilinear Pairing Let $\mathbb{G}_1$ be a multiplicative cyclic group of prime order p. Let g be a generator of $\mathbb{G}_1$ and E be a bilinear map defined as $E : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. The bilinear map E has the following properties: (i) Bilinearity: $\forall g_1, g_2 \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p$, it holds that $E(g_1^a, g_2^b) = E(g_1, g_2)^{ab}$. (ii) Non-degeneracy: $E(g, g) \neq 1$. We also assume that $\mathbb{G}_1$ is a bilinear group. That is, the group operation in $\mathbb{G}_1$ and the bilinear map $E : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ are both efficiently computable.

Group Signature Scheme by Boneh and Shacham Our proposed AdHocSign scheme is designed based on Boneh and Shacham's verifier-local group signature scheme [5], which includes the following primitives:

- *BSG_KeyGen(n).* The primitive outputs a group public key gpk, user private keys $(gsk[1], \cdots, gsk[n])$, and user revocation tokens $grt = (grt[1], \cdots, grt[n])$.
- *BSG_Sign(gpk,gsk[i],M).* The primitive outputs a signature σ of message M for user i who owns private key $gsk[i]$.
- *BSG_Verf(gpk,RL,σ,M).* The primitive verifies whether σ is a valid signature of message M signed by any user whose membership has not been revoked (i.e., the revocation key of the user is not in the revocation list RL).
- *BSG_Trace(M,σ,grt).* Given a message M, a signature σ of the message and the vector of revocation tokens grt, the primitive outputs the index of the user who generated the signature.

AdHocSign Primitives The AdHocSign scheme provides the following primitives:

- *Setup*. The primitive chooses system parameters.
- *AttributeInit(a)*. The primitive chooses and outputs the secrets $\mathbb{S}[a]$ for attribute a.
- *UserInit(e,$\mathbb{A}[e]$,$\mathbb{S}$)*. The primitive takes as inputs an user ID e, the set of attributes $\mathbb{A}[e]$ owned by the user, and the set of secrets $\mathbb{S}$ for all attributes. It outputs the private key gsk_e of the user.
- *AccessStructureInit(T,$\mathbb{S}$)*. It takes as inputs an access structure T and the set of secretes $\mathbb{S}$ for all attributes. It outputs the public key gpk_T regarding T.
- *Sign(gpk_T,gsk_e,M)*. This primitive takes as inputs the public key gpk_T, a private key gsk_e of a certain user e, and a message M. It outputs signature $\sigma_{M,T}$, or $NULL$ if the attributes owned by the user do not satisfy T.
- *Revoke(gpk_T,gsk_e)*. The primitive is called to get the revocation token of user e regarding access structure T. It takes as inputs the public key gpk_T and the private key gsk_e, and outputs the revocation token $grt_T[e]$.
- *Verf(gpk_T,RL,σ,M)*. The primitive takes as inputs the public key gpk_T, list RL of revocation tokens, a signature σ and message M. It outputs *valid* if and only if σ is a valid signature of M regarding T that was generated by a user not revoked.
- *Trace(M,σ,grt_T)*. This primitive takes as inputs M, σ and the set of revocation tokens grt_T. It outputs the ID of the user who generated σ.

Security Assumptions The AdHocSign scheme is designed based on the assumptions about the hardness of the following problems. (i) q-Strong Diffie-Hellman (q-SDH) Problem in $\mathbb{G}_1$: Given a $(q+1)$-tuple g, g^x, $g^{(x^2)}$, $\cdots$, $g^{(x^q)}$ of group $\mathbb{G}_1$ as input, output a pair $(c, g^{1/(x+c)})$ where $c \in \mathbb{Z}_p$. We say that the (q,t,ϵ)-SDH assumption holds in $\mathbb{G}_1$ if no t-time algorithm has the probability of at least ϵ in solving the q-SDH problem in $\mathbb{G}_1$. (ii) Decision Linear Problem in $\mathbb{G}_1$: Given u, v, h, u^a, v^b, $h^c \in \mathbb{G}_1$ as input, output yes if $a+b=c$ and no otherwise. We say that the (t,ϵ)-Decision Linear assumption holds in $\mathbb{G}_1$ if no t-time algorithm has probability of at least $\epsilon + 1/2$ in solving the Decision Linear problem in $\mathbb{G}_1$.

3 Construction for Conjunction-only Access Structures

In this section, we present the design of AdHocSign when the access structure is a conjunction-only logical expression of attributes as $T = a_{T,1} \wedge \cdots \wedge a_{T,s}$, where each $a_{T,i}$ $(i = 1, \cdots, s)$ is an attribute.

3.1 Algorithms

The following algorithms implement the primitives defined in Section 2.

CO_Setup. The setup algorithm chooses the following system parameters: (i) a finite field of integers $\mathbb{Z}_p$ where p is a large prime number, (ii) a multiplicative cyclic and bilinear group $\mathbb{G}_1$ of prime order p, (iii) bilinear map E: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, (iv) a generator g of group $\mathbb{G}_1$, and (v) a secret element $\gamma \in \mathbb{Z}_p$.

CO_AttributeInit(a). The attribute initialization algorithm takes as input an attribute a, and outputs a secret number denoted as α_a, where α_a is picked from $\mathbb{Z}_p$ uniformly at random. α_a is also stored at vector $\mathbb{S}$ at index a; i.e., $\mathbb{S}[a] \leftarrow \alpha_a$.

CO_UserInit($e, \mathbb{A}[e], \mathbb{S}$). The user initialization algorithm takes the following inputs: (i) user ID e, (ii) the set $\mathbb{A}[e] = \{a_{e,i}|i = 1, \cdots, n_e\}$ of attributes assigned to user e, and (iii) the set $\mathbb{S}$ of secret numbers for attributes. The algorithm outputs a private key of the user, i.e., $gsk_e = \langle x_e, \{(a_{e,i}, A_{e,a_{e,i}})|i = 1, \cdots, n_e\}\rangle$, where x_e is picked from $\mathbb{Z}_p$ uniformly at random and $A_{e,a_{e,i}} = g^{\frac{\alpha_{a_{e,i}}}{\gamma + x_e}}$.

CO_AccessStructureInit(T, $\mathbb{S}$). The algorithm takes as inputs the access structure T and the set of secrets $\mathbb{S}$ for all attributes. It picks $r_{a_{T,i}}$ ($i = 1, \cdots, s$) from $\mathbb{Z}_p$ uniformly at random, for each attribute $a_{T,i}$ present in T. Then, it computes and outputs the public key $gpk_T = \langle T, \{r_{a_{T,1}}, \cdots, r_{a_{T,s}}\}, g_T, w_T\rangle$, where $g_T = g^{\sum_{k=1}^{s}(r_{a_{T,k}} \cdot \alpha_{a_{T,k}})}$ and $w_T = g_T^{\gamma}$.

CO_Sign(gpk_T,gsk_e,M). The inputs of the signature generation algorithm include (i) gpk_T: the public key regarding access structure T for which a signature is to be generated; (ii) gsk_e: the private key held by user e for whom the signature is to be generated; and (iii) M: the message to be signed.

The algorithm outputs $\sigma_{M,T}$, signature of message M regarding T. The signature is generated in the following steps: If user e does not have all attributes present in T, $NULL$ is returned. Otherwise, the following steps are executed. (i) $\hat{A}_{e,T} \leftarrow \prod_{i=1}^{s} A_{e,a_{T,i}}^{r_{a_{T,i}}}$; (ii) $\sigma_{M,T} \leftarrow BSG\_Sign(\{g_T, w_T\}, \{\hat{A}_{e,T}, x_e\}, M)$. Here, BSG_Sign is the signing primitive in the group signature scheme proposed by Boneh and Shacham, $\{g_T, w_T\}$ is the public key, and $\{\hat{A}_{e,T}, x_e\}$ is the user private key.

CO_Revoke(gpk_T,gsk_e). This algorithm takes as inputs the public key gpk_T for a certain access structure T and the private key gsk_e for a certain user e. It computes the revocation token of e regarding T as $grt_T[e] \leftarrow \prod_{i=1}^{s} A_{e,a_{T,i}}^{r_{a_{T,i}}}$. Then, the algorithm returns $grt_T[e]$.

CO_Verf(gpk_T,RL_T,σ,M). To verify if σ is a signature of message M regarding access structure T, the verification algorithm takes the public key gpk_T and the list RL_T of revocation keys as inputs. The algorithm can be implemented by calling $BSG\_Verf(\{g_T, w_T\},\ RL_T,\ \sigma,\ M)$. Here, BSG_Verf is the revocation primitive in the group signature scheme proposed by Boneh and Shacham.

CO_Trace(M, σ, grt_T). It can be implemented by calling $BSG\_Trace(M, \sigma, grt_T)$.

3.2 Security Analysis

The security properties of our design are stated in the following theorems, which are proved in [12].

Theorem 1. *If a user e has all attributes in a conjunction-only access structure T, then*

$$\{CO\_Verf(gpk_T, RL_T, CO\_Sign(gpk_T, gsk_e, M), M) = valid\} \Leftrightarrow \{grt_T[e] \notin RL_T\}.$$

Theorem 2. *If the (q, t', ϵ')-SDH assumption holds in $\mathbb{G}_1$, the AdHocSign scheme for conjunction-only access structures is $(t, q_H, q_S, n, m, \epsilon)$-traceable, where $n = q - 1$, $\epsilon = 8n\sqrt{\epsilon' q_H} + 2n/p$, and $t' = \Theta(1) \cdot (t + m \cdot q)$.*

Theorem 3. *The AdHocSign scheme for conjunction-only access structures is (t, q_H, q_S, n, m, ϵ)-selflessly-anonymous assuming the (t', ϵ') Decision Linear assumption holds in group $\mathbb{G}_1$ for $\epsilon' = \frac{\epsilon}{2}(\frac{1}{n^2} - \frac{q_S q_H}{p})$ and $t' = \Theta(1) \cdot (t + mn)$.*

4 Construction for General Access Structures

To lay the foundation for our construction for general access structures, the construction for disjunction-only access structures is presented first, which is then extended to the construction for general access structures.

4.1 Construction for Disjunction-only Access Structures

A disjunction-only access structure can be represented as $T = a_{T,1} \vee \cdots \vee a_{T,s}$, where each $a_{T,i}$ $(i = 1, \cdots, s)$ is an attribute.

DO_Setup. The algorithm is the same as CO_Setup, except that (i) a secret ξ is also randomly chosen from $\mathbb{Z}_p$, and (ii) hash functions $H_1 : \mathbb{Z}_p \times \mathbb{Z}_p \times \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{G}_1$ and $H_2 : \mathbb{Z}_p \times \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{Z}_p$ are chosen.

DO_AttributeInit(a, N). The algorithm takes two inputs: a which is ID of the attribute, and N which is an integer. The algorithm outputs N secret numbers to be associated with attribute a, denoted as $\alpha_{a,i}$ for $i = 1, \cdots, N$, where each $\alpha_{a,i}$ is picked from $\mathbb{Z}_p$ uniformly at random. All these secret numbers are recorded in vector $\mathbb{S}[a]$.

DO_UserInit(e, $\mathbb{A}[e]$, $\mathbb{S}$). The inputs to the algorithm include the user ID e, the set of attributes $\mathbb{A}[e] = \{a_{e,i} | i = 1, \cdots, n_e\}$ owned by the user, and the set of secret keys $\mathbb{S} = \{\alpha_{a_{e,i},j} | i = 1, \cdots, n_e; j = 1, \cdots, N\}$ for the above attributes. The algorithm outputs the private key for user e, i.e., $gsk_e = \langle\, x_e,\ \{\ (a_{e,i},\ A'_{e,a_{e,i},1}, \cdots, A'_{e,a_{e,i},N}) \ |i = 1, \cdots, n_e\ \}\ \rangle$, where x_e is picked from $\mathbb{Z}_p$ uniformly at random, $A'_{e,a_{e,i},j} = A_{e,a_{e,i},j} \cdot H_1(e, a_{e,i}, j, H_2(\xi, a_{e,i}, j))^{-1}$ for $j = 1, \cdots, N$, and $A_{e,a_{e,i},j} = g^{\frac{\alpha_{a_{e,i},j}}{\gamma + x_e}}$.

DO_AccessStructureInit(T, $\mathbb{S}$). The inputs to the algorithm include the access structure T and the set of secret keys $\mathbb{S}$ for all attributes. The algorithm picks r_T from $\mathbb{Z}_p$ uniformly at random. For each $a_{T,i}$, the algorithm also picks $\delta_{T,i}$ from $\{1, \cdots, N\}$ such that $\alpha_{a_{T,i},\delta_{T,i}}$ is a secret associated with $a_{T,i}$ that has not been used in access structure initialization before. If such $\delta_{T,i}$ cannot be found successfully, the primitive fails. Otherwise, the algorithm computes and outputs the following public key regarding T: $gpk_T = \langle\, T, \tilde{R}_T, g_T = g^{r_T}, w_T = g_T^{\gamma}\, \rangle$, where $\tilde{R}_T = \{(\ r_{T,i} = \frac{r_T}{\alpha_{a_{T,i},\delta_{T,i}}}, \delta_{T,i}, h_{T,i} = H_2(\xi, a_{T,i}, \delta_{T,i})\ ) | \ i = 1, \cdots, s\}$.

DO_Sign(gpk_T, gsk_e, M). The signature generation algorithm takes as inputs public key gpk_T regarding T, private key gsk_e of user e, and message M. It outputs signature $\sigma_{M,T}$ on message M regarding T for user e as follows: If user e does not have any attribute appearing in T, $NULL$ is returned. Otherwise, assuming the user owns attribute $a_{T,i}$, the user computes $A_{e,a_{T,i},\delta_{T,i}} = (A'_{e,a_{T,i},\delta_{T,i}}) \cdot H_1(e, a_{T,i}, \delta_{T,i}, h_{T,i})$ and $\hat{A}_{e,T} = A^{r_{T,i}}_{e,a_{T,i},\delta_{T,i}}$, and returns $\sigma_{M,T} = BSG\_Sign(\{g_T, w_T\}, \{\hat{A}_{e,T}, x_e\}, M)$.

DO_Verify(gpk_T,RL_T,σ,M). The algorithm is the same as CO_Verify.

4.2 Construction for General Access Structures: The Algorithms

Based on the schemes for disjunction-only and conjunction-only access structures, the scheme for general access structures is designed as follows.

G_Setup, G_AttributeInit(a,N), G_UserInit(e, $\mathbb{A}[e]$, $\mathbb{S}$) and G_Verify(gpk_T, RL_T, σ, M). The algorithms are the same as DO_Setup, DO_AttributeInit, DO_UserInit and CO_Verify, respectively.

G_AccessStructureInit($\boldsymbol{T}$, $\mathbb{S}$). The inputs to the algorithm include the access structure T as defined in Equations (1) and (2), and the set of private keys $\mathbb{S}$. For each $DT_i = a_{T,i,1} \vee \cdots \vee a_{T,i,s_i}$, the algorithm picks $r_{T,i}$ from $\mathbb{Z}_p$ uniformly at random. Then, for each $a_{T,i,j}$ that is a part of DT_i, the algorithm finds $\delta_{T,i,j}$ from $\{1, \cdots, N\}$ such that $\alpha_{a_{T,i,j},\delta_{T,i,j}}$ is a secret associated with attribute $a_{T,i,j}$ and has not been used in access structure initialization before, and computes $r_{T,i,j} = \frac{r_{T,i}}{\alpha_{a_{T,i,j},\delta_{T,i,j}}}$. If such $\delta_{T,i,j}$ for $j = 1, \cdots, s_i$ cannot be found in $\{1, \cdots, N\}$ successfully, the primitive fails. Finally, the algorithm computes and outputs the following public key: $gpk_T = \langle\ T, \{\ (r_{T,i,j}, \delta_{T,i,j}, h_{T,i,j})\ |i = 1, \cdots, s;$ for each $i, j = 1, \cdots, s_i\},\ g_T = g^{\sum_{i=1}^{s} r_{T,i}},\ w_T = g_T^{\gamma}\ \rangle$, where $h_{T,i,j} = H_2(\xi, a_{T,i,j}, \delta_{T,i,j})$.

G_Sign($\boldsymbol{gpk_T}$, $\boldsymbol{gsk_e}$, $\boldsymbol{M}$). The algorithm takes as inputs public key gpk_T regarding T, private key gsk_e of user e, and message M. It computes and outputs signature $\sigma_{M,T}$ as follows: If the attributes owned by user e do not satisfy T, $NULL$ is returned. Otherwise, assuming the user owns attribute a_{T,i,k_i} for $i = 1, \cdots, s$, the user computes $\hat{A}_{e,T} = \prod_{i=1}^{s}[A'_{e,a_{T,i,k_i},\delta_{T,i,k_i}} \cdot H_1(x_e, a_{T,i,k_i}, \delta_{T,i,k_i}, h_{T,i,k_i})]^{r_{T,i,k_i}}$ and returns $\sigma_{M,T} = BSG\_Sign(\{g_T, w_T\}, \{\hat{A}_{e,T}, x_e\}, M)$.

G_Revoke($\boldsymbol{gpk_T}$,$\boldsymbol{gsk_e}$) This algorithm takes as inputs the public key gpk_T for a certain access structure T and the private key gsk_e for a certain user e. It computes the revocation token of e regarding T as $grt_T[e] \leftarrow \hat{A}_{e,T}$, where the computation of $\hat{A}_{e,T}$ is the same as step 2 in primitive G_Sign. Then, the algorithm returns $grt_T[e]$.

G_Trace($\boldsymbol{M}$, $\boldsymbol{\sigma}$, $\boldsymbol{grt_T}$). This can be implemented with $BSG\_Trace(M, \sigma, grt_T)$.

4.3 Security Analysis

Theorem 4. *If the attributes owned by a user e satisfy an access structure T as defined in Equation (1), then*

$$\{G\_Verf(gpk_T, RL_T, G\_Sign(gpk_T, gsk_e, M), M) = valid\} \Leftrightarrow \{grt_T[e] \notin RL_T\}.$$

Theorem 5. *If the (q, t', ϵ')-SDH assumption holds in $\mathbb{G}_1$, then the AdHocSign scheme for general access structures is $(t, q_H, q_S, n, m, \epsilon)$-traceable, where $n = q - 1$, $\epsilon = 8n\sqrt{\epsilon' q_H} + 2n/p$, $t' = O(t \cdot m \cdot N)$, and N is the maximum number of secrets associated with each attribute.*

Theorem 6. *The AdHocSign scheme for general access structures is (t, q_H, q_S, n, m, ϵ)-selflessly-anonymous assuming the (t', ϵ') Decision Linear assumption holds in group $\mathbb{G}_1$ for $\epsilon' = \frac{\epsilon}{2}(\frac{1}{n^2} - \frac{q_S q_H}{p})$, where $t' = \Theta(1) \cdot (t + m \cdot n \cdot N)$, and N is the maximum number of secrets associated with each attribute.*

The theorems are proved in [12].

5 Related Work

Attribute based encryption (ABE) [1,8,3] provides fine-grained access control to shared data. With ABE, data are encrypted based on their access structures, where an access structure is a logical expression over attributes. A user is able to decrypt a piece of data if and only if she owns the access attributes that satisfy the access policy. However, ABE does not provide accountability. Though it ensures that only authorized users can access the data, it is hard to trace who have accessed the data.

With group signature schemes [2,5,4,7,6,9] every member of a group can sign a message anonymously on behalf of the group, a verifier of the signature can determine whether the singer is a valid member of the group without knowing the identity of the signer, but a trusted authority is able to trace out the identity of the signer when necessary. Different from these works, our proposed AdHocSign is a new group signature scheme that not only inherits the features of accountable and anonymous signature generation/verification, but also enables on-the-fly formation of new groups according to a newly defined access structure (policy). If the AdHocSign is used together with ABE, accountable attribute-based fine-grained access control can be realized.

Khader proposed attribute based group signature schemes [10,11], with a similar design goal as that of AdHocSign. However, a signer in the schemes should inform the verifier of the attributes he/she owns for signature verification. On the contrary, AdHocSign does not let the verifier know what attributes are owned by a singer (unless the access structure is a conjunction-only expression of attributes), and therefore provides more privacy preservation.

6 Conclusion

We have presented a new group signature scheme for dynamically formed groups, to support accountable and anonymous access to outsourced Data. Rigorous security analysis of the scheme has been conducted to prove its selfless-anonymity and traceability based on the hardness assumption of q-SDH and Decisional Linear problems.

References

1. Fuzzy identity-based encryption. In: EUROCRYPT. pp. 457–473 (2005)
2. Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A practical and provably secure coalition-resistant group signature scheme. In: CRYPTO. pp. 255–270 (2000)
3. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE S&P. pp. 321 –334 (2007)
4. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: CRYPTO. pp. 41–55 (2004)
5. Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: ACM CCS. pp. 168–177 (2004)
6. Camenisch, J., Groth, J.: Group signatures: Better efficiency and new theoretical aspects. In: SCN. pp. 120–133 (2004)
7. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: CRYPTO. pp. 56–72 (2004)
8. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: ACM CCS. pp. 89–98 (2006)
9. Groth, J.: Fully anonymous group signatures without random oracles. In: ASIACRYPT. pp. 164–180 (2007)
10. Khader, D.: Attribute based group signature. In: Cryptology ePrint Archive, Report 2007/159 (2007), http://eprint.iacr.org/
11. Khader, D.: Attribute based group signature with revocation. In: Cryptology ePrint Archive, Report 2007/241 (2007), http://eprint.iacr.org/
12. Zhang, W., Wang, C.: An ad hoc group signature scheme for accountable and anonymous access to outsourced data (2012), http://www.cs.iastate.edu/~wzhang/papers/adhocsign.pdf

The security impact of a new cryptographic library

Daniel J. Bernstein[1], Tanja Lange[2], and Peter Schwabe[3]

[1] Department of Computer Science
University of Illinois at Chicago, Chicago, IL 60607–7053, USA
`djb@cr.yp.to`

[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600MB Eindhoven, the Netherlands
`tanja@hyperelliptic.org`

[3] Research Center for Information Technology Innovation and
Institute of Information Science
Academia Sinica
No. 128 Academia Road, Section 2, Nankang, Taipei 11529, Taiwan
`peter@cryptojedi.org`

Abstract. This paper introduces a new cryptographic library, NaCl, and explains how the design and implementation of the library avoid various types of cryptographic disasters suffered by previous cryptographic libraries such as OpenSSL.

Keywords: confidentiality, integrity, simplicity, speed, security

1 Introduction

For most cryptographic operations there exist widely accepted standards, such as the Advanced Encryption Standard (AES) for secret-key encryption and 2048-bit RSA for public-key encryption. These primitives have been extensively studied, and breaking them is considered computationally infeasible on any existing computer cluster.

For each of these cryptographic primitives there exist various implementations and software libraries, and it has become common best practice in the development of secure systems to use the implementations in these libraries as building blocks. One should thus expect that the cryptographic layer of modern

This work was supported by the National Science Foundation under grant 1018836; by the European Commission through the ICT Programme under Contract ICT-2007-216499 CACE and Contract ICT-2007-216676 ECRYPT II; and by the National Science Council, National Taiwan University and Intel Corporation under Grant NSC99-2911-I-002-001 and 99-2218-E-001-007. Part of this work was carried out when Peter Schwabe was employed by National Taiwan University; part of this work was carried out when Peter Schwabe was employed by Technische Universiteit Eindhoven. Permanent ID of this document: `5f6fc69cc5a319aecba43760c56fab04`. Date: 2012.04.24.

information systems does not expose any vulnerabilities to attackers. Unfortunately this expectation is far from reality, as demonstrated by one embarrassing cryptographic failure after another.

A new cryptographic library: NaCl. To address the underlying problems we have designed and implemented a new cryptographic library. The library name, NaCl, is pronounced "salt" and stands for "Networking and Cryptography Library". This paper discusses only the cryptographic part of NaCl; the networking part is still in prototype form.

NaCl is in the public domain and is available from `http://nacl.cr.yp.to` and `http://nacl.cace-project.eu`, along with extensive documentation. The signature component of NaCl is integrated only into the latest development version, which is not yet online, but the same code is available separately as part of the SUPERCOP benchmarking package at `http://bench.cr.yp.to`. NaCl steers clear of all patents that we have investigated and has not received any claims of patent infringement.

The first announcement of NaCl was in 2008. We considered changing the name of the project when Google announced Native Client, but decided that there was no real risk of confusion. The first release of NaCl was in 2009 but was missing some of the important features discussed in this paper; the C++ NaCl API was not released until 2010, for example, and signatures were not released until 2011.

A research paper on cryptographic software normally focuses on optimizing the choice and implementation of a single cryptographic primitive at a specified security level: for example, [11] reports speed records for signatures at a 2^{128} security level. This paper is different. Our goal is to analyze the real-world security benefits of switching from an existing cryptographic library such as OpenSSL [29] to a completely new cryptographic library. Some of these security benefits are tied to performance, as discussed later, so we naturally use the results of papers such as [11]; but what is new in this paper is the security analysis.

Credits. Several of the implementations used in NaCl are partially or entirely from third parties. The portability of NaCl relies on the `ref` implementation of Curve25519 written by Matthew Dempsky (Mochi Media, now Google). From 2009 until 2011 the speed of NaCl on common Intel/AMD CPUs relied on the `donna` and `donna_c64` implementations of Curve25519 written by Adam Langley (Google) — which, interestingly, also appear in Apple's acknowledgments [4] for iOS 4. The newest implementations of Curve25519 and Ed25519 were joint work with Niels Duif (Technische Universiteit Eindhoven) and Bo-Yin Yang (Academia Sinica). The `core2` implementation of AES was joint work with Emilia Käsper (Katholieke Universiteit Leuven, now Google).

Prototype Python wrappers around C NaCl have been posted by Langley; by Jan Mojzis; and by Sean Lynch (Facebook). We will merge these wrappers and integrate them into the main NaCl release as a single supported Python NaCl, in the same way that we support C++ NaCl.

2 The NaCl API

The reader is assumed to be familiar with the fact that most Internet communication today is cryptographically unprotected. The primary goal of NaCl is to change this: to cryptographically protect every network connection, providing strong confidentiality, strong integrity, and state-of-the-art availability against attackers sniffing or modifying network packets.

Confidentiality is limited to packet contents, not packet lengths and timings, so users still need anti-traffic-analysis tools: route obfuscators such as Tor [37], timing obfuscators, etc. Of course, users also need vastly better software security in operating systems, web browsers, document viewers, etc. Cryptography is only one part of security.

This section introduces the functions provided by NaCl, with an emphasis on the simplicity of these functions: more precisely, the simplicity that these functions bring to cryptographic applications. This section is meant mostly as background for the security analysis in subsequent sections, but there are enough differences between the NaCl API and previous APIs to justify a discussion of the details.

The `crypto_box` API. The central job of a cryptographic library is **public-key authenticated encryption**. The general setup is that a sender, Alice, has a packet to send to a receiver, Bob. Alice scrambles the packet using Bob's public key and her own secret key. Bob unscrambles the packet using Alice's public key and his own secret key. "Encryption" refers to confidentiality: an attacker monitoring the network is unable to understand the scrambled packet. "Authenticated" refers to integrity: an attacker modifying network packets is unable to change the packet produced by Bob's unscrambling. (Availability, to the extent that it is not inherently limited by network resources, is provided by higher-level networking protocols that retransmit lost packets.)

A typical cryptographic library uses several steps to authenticate and encrypt a packet. Consider, for example, the following typical combination of RSA, AES, etc.:

- Alice generates a random AES key.
- Alice uses the AES key to encrypt the packet.
- Alice hashes the encrypted packet using SHA-256.
- Alice reads her RSA secret key from "wire format."
- Alice uses her RSA secret key to sign the hash.
- Alice reads Bob's RSA public key from wire format.
- Alice uses Bob's public key to encrypt the AES key, hash, and signature.
- Alice converts the encrypted key, hash, and signature to wire format.
- Alice concatenates with the encrypted packet.

Often even more steps are required for storage allocation, error handling, etc.

NaCl gives Alice a simple high-level `crypto_box` function that does everything in one step, putting a packet into a box that is protected against espionage and sabotage:

```
c = crypto_box(m,n,pk,sk)
```

The function takes the sender's secret key `sk` (32 bytes), the recipient's public key `pk` (also 32 bytes), a packet `m`, and a nonce `n` (24 bytes), and produces an authenticated ciphertext `c` (16 bytes longer than `m`). All of these objects are C++ `std::string` variables, represented in wire format as sequences of bytes suitable for transmission; the `crypto_box` function automatically handles all necessary conversions, initializations, etc. Bob's operation is just as easy, with the keys and packets reversed, using his secret key, Alice's public key, and the same nonce:

```
m = crypto_box_open(c,n,pk,sk)
```

Each side begins with

```
pk = crypto_box_keypair(&sk)
```

to generate a secret key and a public key in the first place.

These C++ functions are wrappers around C functions; the C functions can also be used directly by C applications. The C NaCl API has the same function names but more arguments: for example, `std::string m` is replaced by `unsigned char *m` and `unsigned long long mlen`, and `std::string c` is replaced by `unsigned char *c`. The formats of `m` and `c` in the C NaCl API are padded so that `clen` matches `mlen`, removing the need to pass `clen` explicitly and allowing ciphertexts to be stored on top of plaintexts. Failures are indicated by exceptions in C++ NaCl and a `-1` return value in C NaCl.

Validation of the API. The API described above might seem *too* simple to support the needs of real-world applications. We emphasize that NaCl has already been integrated into high-security applications that are running on the Internet today.

DNSCurve [9], designed by the first author, provides high-security authenticated encryption for Domain Name System (DNS) queries between a DNS resolver and a DNS server. (The server's public key is provided by its parent DNS server, which of course also needs to be secured; the client's public key is provided as part of the protocol.) NaCl has been used successfully for several independent DNSCurve implementations, including an implementation used [18] by the OpenDNS resolvers, which handle billions of DNS queries a day from millions of computers and automatically use DNSCurve for any DNSCurve server. OpenDNS has also designed and deployed DNSCrypt, a variant of DNSCurve that uses NaCl to authenticate and encrypt DNS queries from a DNS *client* to a DNS resolver; two months after the introduction of DNSCrypt, [39] stated that DNSCrypt was already in use by tens of thousands of clients. Other applications of NaCl so far include the QuickTun VPN software [32]; the Ethos operating system [34]; and the first author's prototype implementation of CurveCP [10], a high-security cryptographic version of TCP.

C NaCl allows `crypto_box` to be split into two steps, `crypto_box_beforenm` and `crypto_box_afternm`, slightly compromising simplicity but gaining extra speed as discussed in Section 4. The `beforenm` step preprocesses `pk` and `sk`, preparing to handle any number of messages; the `afternm` step handles `n` and `m`. Most applications actually use this two-step procedure.

Nonces. The `crypto_box` API leaves nonce generation to the caller. This is not meant to suggest that nonce generation is not part of the cryptographer's job; on the contrary, we believe that cryptographers should take responsibility not just for nonces but also for other security aspects of high-level network protocols. The exposure of nonces simply reflects the fact that nonces are integrated into high-level protocols in different ways.

It might seem simplest to always generate a random 24-byte nonce `n`, and to transmit this nonce as part of the authenticated ciphertext; 24-byte random strings have negligible chance of colliding. If ciphertexts are long then one can tolerate the costs of generating this randomness and of expanding each ciphertext by 24 bytes. However, random nonces do nothing to stop the simplest type of forgery, namely a replay. One standard strategy to prevent replays is to include an increasing number in each packet and to reject any packet whose number is not larger than the number in the last verified packet; using these sequence numbers as nonces is simpler than giving each packet a number *and* a random nonce. On the other hand, choosing public nonces as sequence numbers means giving away traffic information that would otherwise be somewhat more expensive for an attacker to collect. Several different solutions appear in the literature; constraints on nonce generation are often tied directly to questions of the security and privacy that users expect.

Current applications of NaCl, such as DNSCurve and CurveCP, have different requirements regarding nonces, replays, forward secrecy, and many other security issues at a higher level than the `crypto_box` API. A nonceless API would require higher-level complications in all of these applications, and would not simplify their security analysis.

The `crypto_sign` API. Sometimes confidentiality is irrelevant: Alice is sending a public message to many people. In this situation it is helpful for a cryptographic library to provide **public-key signatures**: Alice scrambles the message using her own secret key, and Bob unscrambles the message using Alice's public key. Alice's operations are independent of Bob, allowing the scrambled message to be broadcast to any number of receivers. Signatures also provide non-repudiation, while authenticators are always repudiable.

NaCl provides simple high-level functions for signatures: Alice uses

```
pk = crypto_sign_keypair(&sk)
```

to generate a key pair (again 32 bytes for the public key but 64 bytes for the secret key), and

```
sm = crypto_sign(m,sk)
```

to create a signed message (64 bytes longer than the original message). Bob uses

```
m = crypto_sign_open(sm,pk)
```

to unscramble the signed message, recovering the original message.

Comparison to previous work. NaCl is certainly not the first cryptographic library to promise a simple high-level API. For example, Gutmann's cryptlib library [21] advertises a "high-level interface" that "provides anyone with the ability to add strong security capabilities to an application in as little as half an hour, without needing to know any of the low-level details that make the encryption or authentication work." See [22, page 1].

There are, however, many differences between high-level APIs, as illustrated by the following example. The first code segment in the cryptlib manual [22, page 13] ("the best way to illustrate what cryptlib can do") contains the following six function calls, together with various comments:

```
cryptCreateEnvelope( &cryptEnvelope, cryptUser,
   CRYPT_FORMAT_SMIME );
cryptSetAttributeString( cryptEnvelope,
   CRYPT_ENVINFO_RECIPIENT,
   recipientName, recipientNameLength );
cryptPushData( cryptEnvelope, message, messageSize,
   &bytesIn );
cryptFlushData( cryptEnvelope );
cryptPopData( cryptEnvelope, encryptedMessage, encryptedSize,
   &bytesOut );
cryptDestroyEnvelope( cryptEnvelope );
```

This sequence has a similar effect to NaCl's

```
c = crypto_box(m,n,pk,sk)
```

where `message` is the plaintext `m` and `encryptedMessage` is the ciphertext `c`.

The most obvious difference between these examples is in conciseness: cryptlib has separate functions

- `cryptCreateEnvelope` to allocate storage,
- `cryptSetAttributeString` to specify the recipient,
- `cryptPushData` to start the plaintext input,
- `cryptFlushData` to finish the plaintext input,
- `cryptPopData` to extract the ciphertext, and
- `cryptDestroyEnvelope` to free storage,

while NaCl handles everything in one function. The cryptlib program must also call `cryptInit` at some point before this sequence.

A much less obvious difference is in reliability. For example, if the program runs out of memory, NaCl will raise an exception, while the above cryptlib code will fail in unspecified ways, perhaps silently corrupting or leaking data. The cryptlib manual [22, page 35] states that the programmer is required to check that each function returns `CRYPT_OK`, and that the wrong code shown above is included in the manual "for clarity". Furthermore, [22, page 53] says that if messages are large then "only some of the data may be copied in" by `cryptPushData`;

the programmer is required to check `bytesIn` and loop appropriately. Trouble can occur even if messages are short and memory is ample: for example, [22, page 14] indicates that recipient public keys are retrieved from an on-disk database, but does not discuss what happens if the disk fails or if an attacker consumes all available file descriptors.

Some of the differences between these code snippets are really differences between C and C++: specifically, NaCl benefits from C++ exceptions and C++ strings, while cryptlib does not use these C++ features. For applications written in C, rather than C++, the cryptlib API should instead be compared to the C NaCl API:

```
crypto_box(c,m,mlen,n,pk,sk)
```

This C NaCl function cannot raise C++ exceptions, but it also does not need to: its only possible return value is 0, indicating successful authenticated encryption. C NaCl is intended to be usable in operating-system kernels, critical servers, and other environments that cannot guarantee the availability of large amounts of heap storage but that nevertheless rely on their cryptographic computations to continue working. In particular, C NaCl functions do not call `malloc`, `sbrk`, etc. They do use small amounts of stack space; these amounts will eventually be measured by separate benchmarks, so that stack space can be allocated in advance and guaranteed to be adequate.

Perhaps the most important difference between these NaCl and cryptlib examples is that the `crypto_box` output is authenticated and encrypted using keys from Alice and Bob, while the cryptlib output is merely encrypted to Bob without any authentication; cryptlib supports signatures but does not add them without extra programming work. There is a long history of programs omitting cryptographic authentication, incorrectly treating all successfully decrypted data as authentic, and being exploited as a result; with cryptlib, writing such programs is easier than writing programs that include proper authentication. With NaCl, high-security authenticated encryption is the easiest operation.

3 Security

This section presents various case studies of cryptographic disasters, and explains the features of NaCl that eliminate these types of disasters.

Two specific types of disasters are addressed in subsequent sections: Section 4 discusses users deliberately weakening or disabling cryptography to address cryptographic performance problems; Section 5 discusses cryptographic primitives being broken.

No data flow from secrets to load addresses. In 2005, Osvik, Shamir, and Tromer described a timing attack that discovered the AES key of the `dm-crypt` hard-disk encryption in Linux in just 65 milliseconds. See [30] and [38]. The attack process runs on the same machine but does not need any privileges (for example, it can run inside a virtual machine) and does not exploit any kernel software security holes.

This attack is possible because almost all implementations of AES, including the Linux kernel implementation, use fast lookup tables as recommended in the initial AES proposal; see [17, Section 5.2]. The secret AES key inside the kernel influences the table-load addresses, which in turn influence the state of the CPU cache, which in turn influences measurable timings of the attack process; the attack process computes the AES key from this leaked information.

NaCl avoids this type of disaster by systematically avoiding all loads from addresses that depend on secret data. All of the implementations are thus inherently protected against cache-timing attacks. This puts constraints on the implementation strategies used throughout NaCl, and also influences the choice of cryptographic algorithms in NaCl, as discussed in Section 5.

For comparison, Gutmann's cryptlib manual [22, pages 63–64] claims that cache-timing attacks (specifically "observing memory access latencies for cached vs. un-cached data") and branch-timing attacks (see below) provide almost the same "level of access" as "an in-circuit emulator (ICE)" and that there are therefore "no truly effective defences against this level of threat". We disagree. Software side channels on common CPUs include memory addresses and branch conditions but do not include, e.g., the inputs and outputs to a XOR operation; it is well known that the safe operations are adequate in theory to perform cryptographic computations, and NaCl demonstrates that the operations are also adequate in practice. Typical cryptographic code uses unsafe operations, and cache-timing attacks have been repeatedly demonstrated to be effective against such code, but NaCl's approach makes these attacks completely ineffective.

OpenSSL has responded to cache-timing attacks in a different way, not prohibiting secret load addresses but instead using complicated countermeasures intended to obscure the influence of load addresses upon the cache state. This obviously cannot provide the same level of confidence as the NaCl approach: a straightforward code review can convincingly verify the predictability of all load addresses in NaCl, while there is no similarly systematic way to verify the efficacy of other countermeasures. The review of load addresses and branch conditions (see below) can be automated, as explained in [27] and [26], and in fact has already been formalized and automated for large parts of NaCl; see [3] (which comments that "NaCl code follows strict coding policies that make it *formal verification-friendly*" and explains how parts of the code were verified).

No data flow from secrets to branch conditions. Brumley and Tuveri announced in 2011 that they had used a remote timing attack to find the ECDSA private key used for server authentication in a TLS handshake. See [14]. The implementation targeted in this attack is the ECDSA implementation in OpenSSL.

The underlying problem is that most scalar-multiplication (and exponentiation) algorithms involve data flow from secret data into branch conditions: i.e., certain operations are carried out if and only if the key has certain properties. In particular, the OpenSSL implementation of ECDSA uses one of these algorithms. Secret data inside OpenSSL influences the state of the CPU branch unit, which in turn influences the amount of time used by OpenSSL, which in turn influences measurable timings of network packets; the attacker computes the ECDSA key from this leaked information.

NaCl avoids this type of disaster by systematically avoiding all branch conditions that depend on secret data. This is analogous to the prohibition on secret load addresses discussed above; it has pervasive effects on NaCl's implementation strategies and interacts with the cryptographic choices discussed in Section 5.

No padding oracles. In 1998 Bleichenbacher successfully decrypted an RSA-encrypted SSL ciphertext by sending roughly one million variants of the ciphertext to the server and observing the server's responses. The server would apply RSA decryption to each variant and publicly reject the (many) variants not having "PKCS #1" format. Subsequent integrity checks in SSL would defend against forgeries and reject the remaining variants, but the pattern of initial rejections already leaked so much information that Bleichenbacher was able to compute the plaintext. See [13].

NaCl has several layers of defense against this type of disaster:

- NaCl's authenticated-encryption mechanism is designed as a secure unit, always wrapping encryption inside authentication. Nothing is decrypted unless it first survives authentication, and the authenticator's entire job is to prevent the attacker from forging messages that survive authentication.
- Forged messages always follow the same path through authenticator verification, using constant time (depending only on the message length, which is public) and then rejecting the message, with no output other than the fact that the message is forged.
- Even if the attacker forges a variant of a message by sheer luck, the forgery will be visible only through the receiver accepting the message, and standard nonce-handling mechanisms in higher-level protocols will instantly reject any further messages under the same nonce. NaCl derives new authentication and encryption keys for each nonce, so the attacker will have no opportunity to study the effect of those keys on any further messages.

Note that the third defense imposes a key-agility requirement on the underlying cryptographic algorithms.

Most cryptographic libraries responded to Bleichenbacher's attack by trying to hide different types of message rejection, along the lines of the second defense; for example, [23] shows that this approach was adopted by the GnuTLS library in 2006. However, typical libraries continue to show small timing variations, so this defense by itself is not as confidence-inspiring as using strong authentication to shield decryption. Conceptually similar attacks have continued to plague cryptographic software, as illustrated by the SSH attack in [1] in 2009 and the very recent DTLS attack in [2].

Centralizing randomness. In 2006 a Debian developer removed a critical line of randomness-generation code from the OpenSSL package shipped with Debian GNU/Linux. Code-verification tools had complained that the line was producing unpredictable results, and the developer did not see why the line was necessary. Until this bug was discovered in 2008 (see [33]), OpenSSL keys generated under

Debian and Ubuntu were chosen from a set of size only 32768. Breaking the encryption or authentication of any communication secured with such a key was a matter of seconds.

NaCl avoids this type of disaster by simply reading bytes from the operating-system kernel's cryptographic random-number generator. Of course, the relevant code in the kernel needs to be carefully written, but reviewing that code is a much more tractable task than reviewing all of the separate lines of randomness-generation code in libraries that decide to do the job themselves. The benefits of code minimization are well understood in other areas of security; we are constantly surprised by the amount of unnecessary complexity in cryptographic software.

A structural deficiency in the `/dev/urandom` API provided by Linux, BSD, etc. is that using it can fail, for example because the system has no available file descriptors. In this case NaCl waits and tries again. We recommend that operating systems add a reliable `urandom(x,xlen)` system call.

Avoiding unnecessary randomness. Badly generated random numbers were also involved in the recent collapse of the security system of Sony's PlayStation 3 gaming console. Sony used the standard elliptic-curve digital-signature algorithm, ECDSA, but ignored the ECDSA requirement of a new random secret for each message: Sony simply used a constant value for all messages. Attackers exploited this mistake to compute Sony's root signing key, as explained in [15, slides 122–130], breaking the security system of the PlayStation 3 beyond repair.

NaCl avoids this type of disaster by using *deterministic* cryptographic operations to the extent possible. The `keypair` operations use new randomness, but all of the other operations listed above produce outputs determined entirely by their inputs. Of course, this imposes a constraint upon the underlying cryptographic primitives: primitives that use randomness, such as ECDSA, are rejected in favor of primitives that make appropriate use of pseudorandomness.

Determinism also simplifies testing. NaCl includes a battery of automated tests shared with eBACS (ECRYPT Benchmarking of Cryptographic Systems), an online cryptographic speed-measurement site [12] designed by the first two authors; this site has received, and systematically measured, 1005 implementations of various cryptographic primitives from more than 100 people. The test battery found, for example, that software for a cipher named Dragon was sometimes reading outside its authorized input arrays; the same software had passed previous cryptographic test batteries. All of the core NaCl functions have also been tested against pure Python implementations, some written ourselves and some contributed by Matthew Dempsky.

4 Speed

Cryptographic performance problems have frequently caused users to reduce their cryptographic security levels or to turn off cryptography entirely. Consider the role of performance in the following examples:

- `https://sourceforge.net/account` is protected by SSL, but `https://sourceforge.net/develop` redirects the user's web browser to `http://sourceforge.net/develop`, actively *turning off SSL* and exposing the web pages to silent modification by sniffing attackers. Cryptography that is not actually used can be viewed as the ultimate disaster, providing no more security than any of the other cryptographic disasters discussed in this paper.
- OpenSSL's AES implementations continue to use table lookups on most CPUs, rather than obviously safe bitsliced computations that would be slower on those CPUs. The table lookups have been augmented with several complicated countermeasures that are hoped to protect against the cache-timing attacks discussed in Section 3.
- Google has begun to allow SSL for more and more services, but only with a 1024-bit RSA key, despite
  - recommendations from the RSA company to move up to at least 2048-bit RSA by the end of 2010;
  - the same recommendations from the U.S. government; and
  - analyses from 2003 concluding that 1024-bit RSA was already breakable in under a year using hardware that governments and large companies could already afford.

  See, e.g., [31] for an analysis by Shamir (the S in RSA) and Tromer; [24] for an end-of-2010 recommendation from the RSA company; and [5] for an end-of-2010 recommendation from the U.S. government.
- DNSSEC recommends, and uses, 1024-bit RSA for practically all signatures rather than 2048-bit RSA, DSA, etc.: "In terms of performance, both RSA and DSA have comparable signature generation speeds, but DSA is much slower for signature verification. Hence, RSA is the recommended algorithm. ... The choice of key size is a tradeoff between the risk of key compromise and performance. ... RSA-SHA1 (RSA-SHA-256) until 2015, 1024 bits." See [16].
- The Tor anonymity network [37] also uses 1024-bit RSA.

Speed of NaCl. We do not provide any low-security options in NaCl. For example, we do not allow encryption without authentication; we do not allow any data flow from secrets to load addresses or branch conditions; and we do not allow cryptographic primitives breakable in substantially fewer than 2^{128} operations, such as RSA-2048.

The remaining risk is that users find NaCl too slow and turn it off, replacing it with low-security cryptographic software or no cryptography at all. NaCl avoids this type of disaster by providing exceptionally high speeds. NaCl is generally much faster than previous cryptographic libraries, even if those libraries are asked for lower security levels. More to the point, NaCl is fast enough to handle packet rates beyond the worst-case packet rates of a typical Internet connection.

For example, using a single AMD Phenom II X6 1100T CPU, NaCl performs

- more than 80000 `crypto_box` operations (public-key authenticated encryption) per second;

– more than 80000 `crypto_box_open` operations (public-key authenticator verification and decryption) per second;
– more than 70000 `crypto_sign_open` operations (signature verification) per second; and
– more than 180000 `crypto_sign` operations (signature generation) per second

for any common packet size. (For comparisons to the speeds of other libraries, see, e.g., [11].) To put these numbers in perspective, imagine a connection flooded with 50-byte packets, each requiring a `crypto_box_open`; 80000 such packets per second would consume 32 megabits per second even without packet overhead. A lower volume of network traffic means that the CPU needs only a fraction of its time to handle the cryptography.

NaCl provides even better speeds than this, for four reasons:

– NaCl uses a single public-key operation for a packet of any size, allowing large packets to be handled with very fast secret-key cryptography; 80000 1500-byte packets per second would fill up a gigabit-per-second link.
– A single public-key operation is shared by many packets from the same public key, allowing all the packets to be handled with very fast secret-key cryptography, if the caller splits `crypto_box` into `crypto_box_beforenm` and `crypto_box_afternm`.
– NaCl uses "encrypt-then-MAC", so forged packets are rejected without being decrypted; a flood of forgeries thus has even more trouble consuming CPU time.
– The signature system in NaCl supports fast batch verification, effectively doubling the speed of verifying a stream of valid signatures.

Most of these speedups do not reduce the cost of handling forgeries under *new* public keys, but a flooded server can continue providing very fast service to public keys that are *already known*.

The optimized implementations in the current version of NaCl are aimed at large CPUs, but all of the cryptographic primitives in NaCl can fit onto much smaller CPUs: there are no requirements for large tables or complicated code. NaCl also makes quite efficient use of bandwidth: as mentioned earlier, public keys are only 32 bytes, signed messages are only 64 bytes longer than unsigned messages, and authenticated ciphertexts are only 16 bytes longer than plaintexts.

5 Cryptographic primitives in NaCl

Stevens, Sotirov, Appelbaum, Lenstra, Molnar, Osvik, and de Weger announced in 2008 (see [35] and [36]) that, by exploiting various weaknesses that had been discovered in the MD5 hash function, they had created a rogue CA certificate. They could, if they wanted, have impersonated any SSL site on the Internet.

This type of disaster, cryptographic primitives being broken, is sometimes claimed to be prevented by cryptographic standardization. However, there are many examples of standards that have been publicly broken, including DES, 512-bit RSA, and these MD5-based certificates. More to the point, there are some existing standards that can reach NaCl's speeds, but those standards fall far short of NaCl's security requirements.

Our main strategy for avoiding dangerous primitives in NaCl has been to *pay attention to cryptanalysis*. There is an extensive cryptanalytic literature exploring the limits of attacks on various types of cryptographic primitives; some cryptographic structures are comfortably beyond these limits, while others inspire far less confidence. This type of security evaluation is only loosely related to standardization, as illustrated by the following example: Dobbertin, Bosselaers, and Preneel wrote "It is anticipated that these techniques can be used to produce collisions for MD5 and perhaps also for RIPEMD" in 1996 [19], eight years before collisions in MD5 (and RIPEMD) were published and a decade before most MD5-based standards were withdrawn. They recommended switching to RIPEMD-160, which fifteen years later has still not been publicly broken.

This strategy, choosing cryptographic algorithms in light of the cryptanalytic literature, has given us higher confidence in NaCl's cryptographic primitives than in most standards. At the same time this strategy has given us the flexibility needed to push NaCl to extremely high speeds, avoiding the types of disasters discussed in Section 4.

The rest of this section discusses the cryptographic primitives used in NaCl, and explains why we expect these choices to reduce the risk of cryptographic disasters. Specifically, NaCl uses elliptic-curve cryptography, not RSA; it uses an elliptic curve, Curve25519, that has several advanced security features; it uses Salsa20, not AES (although it does include an AES implementation on the side); it uses Poly1305, not HMAC; and for an elliptic-curve signature system it uses EdDSA, not ECDSA.

We are aware that many existing protocols require AES and RSA, and that taking advantage of NaCl as described in this paper requires those protocols to be upgraded. We have prioritized security over compatibility, and as a consequence have also prioritized speed over compatibility. There are other projects that have explored the extent to which speed and security can be improved without sacrificing compatibility, but NaCl is aiming at a different point in the design space, and at applications that are not well served by the existing protocols. DNSCrypt (see Section 2) illustrates the feasibility of our deployment approach.

Cryptographic choices in NaCl. RSA is somewhat older than elliptic-curve cryptography: RSA was introduced in 1977, while elliptic-curve cryptography was introduced in 1985. However, RSA has shown many more weaknesses than elliptic-curve cryptography. RSA's effective security level was dramatically reduced by the linear sieve in the late 1970s, by the quadratic sieve and ECM in the 1980s, and by the number-field sieve in the 1990s. For comparison, a few attacks have been developed against some rare elliptic curves having special algebraic structures, and the amount of computer power available to attackers has predictably increased, but typical elliptic curves require just as much computer power to break today as they required twenty years ago.

IEEE P1363 standardized elliptic-curve cryptography in the late 1990s, including a stringent list of security criteria for elliptic curves. NIST used the IEEE P1363 criteria to select fifteen specific elliptic curves at five different security levels. In 2005, NSA issued a new "Suite B" standard, recommending the NIST elliptic curves (at two specific security levels) for all public-key cryptography and withdrawing previous recommendations of RSA.

Curve25519, the particular elliptic curve used in NaCl, was introduced in [7] in 2006. It follows all of the standard IEEE P1363 security criteria; it *also* satisfies new recommendations for "twist security" and "Montgomery representation" and "Edwards representation". What this means is that secure implementations of Curve25519 are considerably simpler and faster than secure implementations of (e.g.) NIST P-256; there are fewer opportunities for implementors to make mistakes that compromise security, and mistakes are more easily caught by reviewers.

Montgomery representation allows fast single-scalar multiplication using a Montgomery ladder [28]; this is the bottleneck in Diffie–Hellman key exchange inside `crypto_box`. It was proven in [7] that this scalar-multiplication strategy removes all need to check for special cases inside elliptic-curve additions. NaCl uses a ladder of fixed length to eliminate higher-level branches. Edwards representation allows fast multi-scalar multiplication and general addition with the same advantage of not having to check for special cases. The fixed-base-point scalar multiplication involved in `crypto_sign` uses Edwards representation for additions, and eliminates higher-level branches by using a fixed sequence of 63 point additions as described in [11, Section 4].

Salsa20 [8] is a 20-round 256-bit cipher that was submitted to eSTREAM, the ECRYPT Stream Cipher Project [20], in 2005. The same project collected dozens of submissions from 97 cryptographers in 19 countries, and then hundreds of papers analyzing the submissions. Four refereed papers from 14 cryptographers studied Salsa20, culminating in a 2^{151}-operation "attack" against 7 rounds and a 2^{249}-operation "attack" against 8 rounds. After 3 years of review the eSTREAM committee selected a portfolio of 4 software ciphers, including Salsa20; they recommended 12 rounds of Salsa20 as having a "comfortable margin for security".

For comparison, AES is a 14-round 256-bit cipher that was standardized ten years ago. Cryptanalysis at the time culminated in a 2^{140}-operation "attack" against 7 rounds and a 2^{204}-operation "attack" against 8 rounds. New research in 2011 reported a 2^{254}-operation "attack" against all 14 rounds, marginally exploiting the slow key expansion of AES, an issue that was avoided in newer designs such as Salsa20. (Salsa20 also has no penalty for switching keys.) Overall each round of Salsa20 appears to have similar security to each round of AES, and 20 rounds of Salsa20 provide a very solid security margin, despite being faster than 14 rounds of AES on most CPUs.

A further difficulty with AES is that it relies on lookup tables for high-speed implementations; avoiding lookup tables compromises the speed of AES on most CPUs. Recall that, as discussed in Section 3, NaCl prohibits loading data from secret addresses. We do not mean to say that AES cannot be implemented securely: the NaCl implementation of AES is the bitsliced assembly-language implementation described in [25], together with a portable C implementation following the same approach. However, we are concerned about the extent to which security for AES requires compromising speed. Salsa20 avoids these issues: it avoids all use of lookup tables.

Poly1305 is an information-theoretically secure message-authentication code introduced in [6]. Using Poly1305 with Salsa20 is guaranteed to be as secure as using Salsa20 alone, with a security gap of at most 2^{-106} per byte: an attacker who can break the Poly1305 authentication can also break Salsa20. HMAC does not offer a comparable guarantee.

EdDSA was introduced quite recently in [11]. It is much newer than other primitives in NaCl but is within a well-known cloud of signature systems that includes ElGamal, Schnorr, ECDSA, etc.; it combines the safest choices available within that cloud. EdDSA is like Schnorr and unlike ECDSA in that it diversifies the hash input, adding resilience against hash collisions, and in that it avoids inversions, simplifying and accelerating implementations. EdDSA differs from Schnorr in using a double-size hash function, further reducing the risk of any hash-function problems; in requiring Edwards curves, again simplifying and accelerating implementations; and in including the public key as a further input to the hash function, alleviating concerns regarding attacks targeting many keys at once. EdDSA also avoids a minor compression mechanism, as discussed in [11]; the compression mechanism is public, so it cannot improve security, and skipping it is essential for EdDSA's fast batch verification. Finally, EdDSA generates per-message secret nonces by hashing each message together with a long-term secret, rather than requiring new randomness for each message.

NaCl's implementation of `crypto_sign` *does* use lookup tables but nevertheless avoids secret indices: each lookup from the table loads all table entries and uses arithmetic to obtain the right value. For details see [11, Section 4]. NaCl's signature verification uses signed-sliding-window scalar multiplication, which takes different amounts of time depending on the scalars, but this does not create security problems and does not violate NaCl's prohibition on secret branches: the scalars are not secret.

To summarize, all of these cryptographic choices are quite conservative. We do not expect any of them to be broken until someone succeeds in building a large quantum computer; before that happens we will extend NaCl to support post-quantum cryptography.

References

1. Martin R. Albrecht, Kenneth G. Paterson, and Gaven J. Watson. Plaintext recovery attacks against SSH. In David Evans and Andrew Myers, editors, *2009 IEEE Symposium on Security and Privacy, Proceedings*, pages 16–26. IEEE Computer Society, 2009. http://www.isg.rhul.ac.uk/~kp/SandPfinal.pdf. 3

2. Nadhem J. Alfardan and Kenneth G. Paterson. Plaintext-recovery attacks against datagram TLS. http://www.isg.rhul.ac.uk/~kp/dtls.pdf; NDSS 2012, to appear. 3
3. J. Bacelar Almeida, Manuel Barbosa, Jorge S. Pinto, and Bárbara Vieira. Formal verification of side channel countermeasures using self-composition. *Science of Computer Programming*. http://dx.doi.org/10.1016/j.scico.2011.10.008; to appear. 3
4. Apple. iPhone end user licence agreement. Copy distributed inside each iPhone 4; transcribed at http://rxt3ch.wordpress.com/2011/09/27/iphone-end-user-liscence-agreement-quick-refrence/. 1
5. Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management—part 1: General (revised). NIST Special Publication 800-57, 2007. http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-57Part1_3-8-07.pdf. 4
6. Daniel J. Bernstein. The Poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption*, volume 3557 of *LNCS*, pages 32–49. Springer, 2005. http://cr.yp.to/papers.html#poly1305. 5
7. Daniel J. Bernstein. Curve25519: new Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography—PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, 2006. http://cr.yp.to/papers.html#curve25519. 5
8. Daniel J. Bernstein. The Salsa20 family of stream ciphers. In Matthew Robshaw and Olivier Billet, editors, *New stream cipher designs: the eSTREAM finalists*, volume 4986 of *LNCS*, pages 84–97. Springer, 2008. http://cr.yp.to/papers.html#salsafamily. 5
9. Daniel J. Bernstein. DNSCurve: Usable security for DNS, 2009. http://dnscurve.org/. 2
10. Daniel J. Bernstein. CurveCP: Usable security for the Internet, 2011. http://curvecp.org/. 2
11. Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 124–142. Springer, 2011. http://eprint.iacr.org/2011/368. 1, 4, 5
12. Daniel J. Bernstein and Tanja Lange (editors). eBACS: ECRYPT benchmarking of cryptographic systems. http://bench.cr.yp.to. 3
13. Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1. In Hugo Krawczyk, editor, *Advances in Cryptology—CRYPTO '98*, volume 1462 of *LNCS*, pages 1–12. Springer, 1998. http://www.bell-labs.com/user/bleichen/papers/pkcs.ps. 3
14. Billy Bob Brumley and Nicola Tuveri. Remote timing attacks are still practical. In Vijay Atluri and Claudia Diaz, editors, *Computer Security—ESORICS 2011*, volume 6879 of *LNCS*, pages 355–371. Springer, 2011. http://eprint.iacr.org/2011/232/. 3
15. "Bushing", Hector Martin "marcan" Cantero, Segher Boessenkool, and Sven Peter. PS3 epic fail, 2010. http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf. 3
16. Ramaswamy Chandramouli and Scott Rose. Secure domain name system (DNS) deployment guide. NIST Special Publication 800-81r1, 2010. http://csrc.nist.gov/publications/nistpubs/800-81r1/sp-800-81r1.pdf. 4
17. Joan Daemen and Vincent Rijmen. AES proposal: Rijndael, version 2, 1999. http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf. 3

18. Matthew Dempsky. OpenDNS adopts DNSCurve. http://blog.opendns.com/2010/02/23/opendns-dnscurve/. 2
19. Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In Dieter Gollmann, editor, *Fast Software Encryption*, volume 1039 of *LNCS*, pages 71–82. Springer, 1996. 5
20. ECRYPT. The eSTREAM project. http://www.ecrypt.eu.org/stream/. 5
21. Peter Gutmann. cryptlib security toolkit. http://www.cs.auckland.ac.nz/~pgut001/cryptlib/. 2
22. Peter Gutmann. cryptlib security toolkit: version 3.4.1: user's guide and manual. ftp://ftp.franken.de/pub/crypt/cryptlib/manual.pdf. 2, 3
23. Simon Josefsson. Don't return different errors depending on content of decrypted PKCS#1. Commit to the GnuTLS library, 2006. http://git.savannah.gnu.org/gitweb/?p=gnutls.git;a=commit;h=fc43c0d05ac450513b6dcb91949ab03eba49626a. 3
24. Burt Kaliski. TWIRL and RSA key size. http://web.archive.org/web/20030618141458/http://rsasecurity.com/rsalabs/technotes/twirl.html. 4
25. Emilia Käsper and Peter Schwabe. Faster and timing-attack resistant AES-GCM. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems—CHES 2009*, volume 5747 of *LNCS*, pages 1–17. Springer, 2009. http://cryptojedi.org/papers/#aesbs. 5
26. Adam Langley. ctgrind—checking that functions are constant time with Valgrind, 2010. https://github.com/agl/ctgrind. 3
27. David Molnar, Matt Piotrowski, David Schultz, and David Wagner. The program counter security model: Automatic detection and removal of control-flow side channel attacks. In Dongho Won and Seungjoo Kim, editors, *Information Security and Cryptology: ICISC 2005*, volume 3935 of *LNCS*, pages 156–168. Springer, 2005. 3
28. Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987. http://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866113-7/S0025-5718-1987-0866113-7.pdf. 5
29. OpenSSL. OpenSSL: The open source toolkit for SSL/TLS. http://www.openssl.org/. 1
30. Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of AES. In David Pointcheval, editor, *Topics in Cryptology—CT-RSA 2006*, volume 3860 of *LNCS*, pages 1–20. Springer, 2006. 3
31. Adi Shamir and Eran Tromer. Factoring large numbers with the TWIRL device. In Dan Boneh, editor, *Advances in Cryptology—CRYPTO 2003*, volume 2729 of *LNCS*, pages 1–26. Springer, 2003. http://tau.ac.il/~tromer/papers/twirl.pdf. 4
32. Ivo Smits. QuickTun. http://wiki.ucis.nl/QuickTun. 2
33. Software in the Public Interest, Inc. Debian security advisory, DSA-1571-1 openssl—predictable random number generator, 2008. http://www.debian.org/security/2008/dsa-1571. 3
34. Jon A. Solworth. Ethos: an operating system which creates a culture of security. http://rites.uic.edu/~solworth/ethos.html. 2
35. Alexander Sotirov, Marc Stevens, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. MD5 considered harmful today, 2008. http://www.win.tue.nl/hashclash/rogue-ca/. 5
36. Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short chosen-prefix collision for MD5

and the creation of a rogue CA certificate. In Shai Halevi, editor, *Advances in Cryptology—CRYPTO 2009*, volume 5677 of *LNCS*, pages 55–69. Springer, 2009. `http://eprint.iacr.org/2009/111/`. 5

37. Tor project: Anonymity online. `https://www.torproject.org/`. 2, 4
38. Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient cache attacks on AES, and countermeasures. *Journal of Cryptology*, 23(1):37–71, 2010. 3
39. David Ulevitch. Want to do something that matters? Then read on. `http://blog.opendns.com/2012/02/06/dnscrypt-hackers-wanted/`. 2

RSA modulus generation in the two-party case

Gérald Gavin[1] and François Arnault[2]

[1]ERIC Lab - University of Lyon - 5, avenue Pierre Mendès France
E-mail: `gavin@univ-lyon1.fr`
[2] Département de Mathématiques et Informatique, Laboratoire XLIM Faculté des Sciences et Techniques 123 avenue Albert Thomas, 87060 LIMOGES cedex
E-mail: `arnault@unilim.fr`

Abstract. In this paper, secure two-party protocols are provided in order to securely generate a random k-bit RSA modulus n keeping its factorization secret. We first show that most existing two-party protocols based on Boneh's test are not correct: an RSA modulus can be output in the malicious case. Recently, Hazay et al. [13] proposed the first proven secure protocol against any polynomial active adversary. However, their protocol is very costly: several hours are required to output a 1024-bit RSA modulus on a standard platform. In this paper, we propose an other approach consisting of post-processing efficient existing Boneh's based protocols. The running time of this post-processing can be neglected with respect to the running time of the whole protocol.

Keywords: RSA modulus, Boneh's test, keys share.

1 Introduction

Many cryptographic schemes are based on RSA moduli, for example RSA and Paillier. Many threshold versions have been proposed [20], [19], [11], [9], [7]. In these versions, parties must generate an RSA modulus n and distribute public and private keys such that the factorization of n is kept secret. A solution consists of invoking a third party, called the trusted dealer, which generates RSA moduli and shared keys.

The natural question is to know whether the dealer can be efficiently removed. Boneh and Franklin [3] provided a positive answer when the number of parties is larger than 3. This result is based on an adaptation of the Fermat's primality test for RSA moduli, called Boneh's test. It is shown secure against passive adversaries. Frankel et al. [22], Algesheimer et al. [1] have proposed a robust protocol for the multi-party case in the presence of a minority of arbitrarily misbehaving malicious parties. Damgard and Mikkelsen [10] have proposed an other biprimality test with a better error estimate, but it cannot be used directly in the two-party setting with active adversaries. The security of threshold Paillier's scheme requires that moduli are safe, i.e the product of safe primes. Fouque and Stern [11] have shown that it suffices that the sub-group QR(n) of quadratic residues is cyclic to ensure security. They provide secure multi-party protocols for RSA moduli having this property.

This paper deals with the two-party case. Gilboa [12] Poupard and Stern [18] and Straub [21] have proposed a solution based on Oblivious Transfert (OT). They propose a robust protocol ModulusGeneration which computes $n = (p_A + p_B)(q_A + q_B)$ where p_A, q_A and p_B, q_B are two k-bit integers randomly chosen respectively by Alice and by Bob. They then apply Boneh's test [4]. For concreteness, this biprimality test consists of randomly choosing $a \in Z_n^*$ such that $(\frac{a}{n}) = 1$, then Alice computes $v_A = a^{(n-p_A-q_A+1)/4} \bmod n$ and Bob computes $v_B = a^{(p_B+q_B)/4} \bmod n$. If $v_A/v_B \neq \pm 1 \bmod n$ then the test fails. If the test does not fail for several values of a there is a high probability that n is an RSA modulus. However, let us imagine that n is **not** an RSA modulus and that Alice knows the factorization of n while Bob does not (Alice could have greater resources or better factorization algorithms). In this case, Alice can guess (with non negligible probability) $p_B + q_B$ and thus v_B. She could therefore force Bob to believe that n is an RSA modulus. It proves that this solution is only secure against passive adversary (see appendix A for a concrete presentation and evaluation of this attack). In [2], the authors have proposed solutions against this attack. However their protocols have not been proven secure and above all their solution is not at all practical. For instance, $1,6.10^8$ El Gamal encryptions/decryptions are required to output a 1024-bit RSA modulus. Recently, Hazay et al. [13] proposed the first proven secure protocol against any polynomial active adversary. To reach this security level, authors propose a zero-knowledge proof to prove that v_A and v_B are correctly computed. However, this zero-knowledge proof is executed for each candidate n. This protocol based on a cut-and-choose approach is very costly: it requires $O(l)$ (l being a security parameter) exponentiations per loop in Boneh's test. As the number of candidates n which should be tested is about $O(k^2)$, the number of exponentiation of their protocol is $O(l^2k^2)$. We roughly estimate that several hours are needed to output a 1024-RSA modulus with this protocol (more than 10^7 modular exponentiations are required).

The key idea of this paper consists of post-processing existing (efficient) protocols based on Boneh's test with a running time in $O(l)$ (few seconds are requiring for 1024-bit numbers): this post-processing can be neglected with respect to the whole protocol. This idea was already suggested in [13] (section "Practical considerations") in order to drastically improve efficiency. However, security was not proven in this setting: authors claim: "*We cannot simply postpone all proofs; care must be taken to allow simulation and not to reveal information that would allow a malicious party to, e.g., fake some zero-knowledge proof at a later point*". While our approach differs from theirs, this paper can be seen as a positive answer to this claim.

For concreteness, the output n of classical Boneh's test based protocols is tested again. Here, it is assumed the existence of a correct and private protocol ModulusGeneration computing $n = (p_A + p_B)(q_A + q_B)$ (many versions can be found in the literature [12], [3], [18], [21]). The parties then test whether n is an RSA modulus with the classical Boneh's test (see [4]). As discussed before, this test may be incorrect against active adversaries. Thus, if (and only if) n has been determined an RSA modulus previously, it is tested again with a test directly derived from Boneh's test. This test is implemented in protocols Gamma and RSAModulusTest. The efficiency improvement comes from the fact that this second test is applied only once. The main protocol of this paper, RSAModulusGeneration is an implementation of the following generic scheme:

1. Each party generates a public key and a secret key of an **additively** homomorphic encryption scheme S. E_A (resp. E_B) refers to the encryption function generated by Alice (resp. Bob).
2. They invoke ModulusGeneration to get moduli n until n succeeds the classical Boneh's test. Each time the test fails, each party checks that the other party has behaved honestly.
3. Alice (resp. Bob) invokes Gamma to get an encryption $\Gamma_B = E_B((p-1)(q-1))$ (resp. $\Gamma_A = E_A((p-1)(q-1))$) where $n = pq$
4. Alice (resp. Bob) invokes RSAModulusTest (as party 1) on the input Γ_B (resp. Γ_A) to be convinced that n is really an RSA modulus. If the test fails, the protocol fails (it means that a party has behaved dishonestly in step 2.)

It is important to notice that an adversary does not gain any advantage when RSAModulusGeneration fails. This is used to simplify and improve protocols and their security proof. We propose original tools to prove that an adversary **cannot learn** the factorization of n without making the protocol fails: this is captured in the definition of the property nPrivate introduced in section 2. This property is dedicated to factorization problem but can be straightforwardly extended to other problems.

2 Problem and security statements

Let us make explicit the conjecture intrinsically made in Boneh's test based protocols. This conjecture is related to the difficulty of factorizing RSA moduli.

Conjecture 1. Let $k \in \mathbb{N}$ and $n = pq$ be an RSA modulus chosen at random such that $p \equiv q \equiv 3 \mod 4$, $p \in [p_A + 2^k, p_A + 2^{k+1}[\bigcap \mathbb{N}$ and $q \in [q_A + 2^k, q_A + 2^{k+1}[\bigcap \mathbb{N}$ where p_A, q_A are two arbitrary integers. There exists a negligible function δ such that for all probabilistic polynomial-time (p.p.t) algorithms A, A outputs the factorization of n with a probability smaller than $\delta(k)$ only given n, p_A and q_A.

In this paper we wish to design a two-party protocol for securely generating RSA moduli. Existing protocols for securely computing RSA moduli are decomposed in two parts. First, parties securely compute a candidate $n = pq$ by using a secure protocol called here ModulusGeneration, they then test n with Boneh's test. In this paper, we assume the existence of such a protocol ModulusGeneration whose the specifications are detailed in the next definition. In particular, it should be correct and private (see [14]).

Definition 1. *ModulusGeneration is protocol implementing the following ideal scheme. Alice chooses two secret positive integers p_A, q_A and Bob chooses two secret positive integers p_B, q_B. Then, they send these values to a trusted party which outputs $n = pq$ where $p = p_A + p_B$ and $q = q_A + q_B$ if and only if $p_A \equiv q_A \equiv 3 \mod 4$ and $p_B \equiv q_B \equiv 0 \mod 4$. ModulusGeneration is assumed to be* ***correct and private*** *against any active polynomial adversary.*

ModulusGeneration checks that $p_A + p_B \equiv q_A + q_B \equiv 3 \mod 4$ because it is required for Boneh's test but ModulusGeneration does not need to verify the size of the input integers p_A, q_A, p_B and q_B. It will be done in the protocol Gamma only once, i.e. on the first modulus n which successfully passes Boneh's test. Versions of ModulusGeneration can be found in the literature (see [12], [18]). In section 3, we present a two-party protocol, called RSAModulusGeneration, which securely (we will clarify what it means below) generates RSA moduli. The idea consists of generating an RSA modulus n in the classical way (ModulusGeneration + Boneh's test). If n is determined to be an RSA modulus then it is tested again (Gamma + RSAModulusTest). The second test make RSAModulusGeneration fail if n is not an RSA modulus. Thus, the second test is applied only once. The

RSAModulusGeneration security proof exploits the fact that an adversary A does not gain any advantage when the protocol fails. For this reason, weaker security requirements will be considered by restricting the security analysis of RSAModulusGeneration to the two following issues :

1. RSAModulusGeneration is correct, i.e. the output n is not a k-bit RSA modulus with negligible probability with respect to k.
2. Assuming conjecture 1, for any polynomial adversary A, the probability that the protocol does not fail and that A factors the output n is negligible with respect to k.

In order to prove the second issue, we introduce a property dedicated to our problem, called nPrivate. This property can be interpreted as the difficulty for an adversary to **learn** the factorization of n during the execution of P without making P fail. This composable property is formalized in the next definition. In section 4, we will show that Gamma and RSAModulusTest satisfy this property. Roughly speaking, these protocols are private against adversaries which do not make the protocol fail with non-negligible probability.

Definition 2. *For any $k \in \mathbb{N}$, Γ_k refers to the set of k-bit integers. Let P be an arbitrary two-party protocol where a public integer $n \in \Gamma_k$ is input to P by both parties. Parties may have arbitrary other inputs. Let us assume that the party 1 is honest and the party 2 is controlled by a polynomial adversary A.*

1. *For any $k \in \mathbb{N}$, D_k denotes a probability distribution over $\Gamma_k \times \{0,1\}^*$. A family of such probability distributions $D = (D_k)_{k\in\mathbb{N}}$ is said nPrivate[1] if for any p.p.t algorithm F, the probability that F outputs the factorization of n given $(n, z_n) \leftarrow D_k$ is negligible.*
2. *Let $D = (D_k)_{k\in\mathbb{N}}$ be an arbitrary nPrivate probability distribution family. We propose the following experiment :*
   (a) *a pair $(n, z_n) \in \Gamma_k \times \{0,1\}^*$ is randomly generated according to D_k, i.e. $(n, z_n) \leftarrow D_k$.*
   (b) *The corrupted party receives (n, z_n)*
   (c) *The honest party receives n and arbitrary information about n.*
   (d) *Parties execute P where the honest party inputs n (and arbitrary other inputs).*

[1] Assuming conjecture 1, such distributions exist.

We say that P is nPrivate *against A if the probability that P has not failed and A outputs the factorization of n at the end of the previous experiment is negligible*

This definition is maybe difficult to understand because it is not only a security definition but it also depends on the functionality computed by P: for instance, P is not nPrivate against passive adversary if the factorization of n is expected to be output. On the other hand, classical Boneh's test is trivially nPrivate meaning that if an RSA modulus is tested, an adversary cannot learn the factorization of n without cheating the test (even by making it fail). Typically, the vector z_n contains the information learnt during the previous protocols. For instance, before the execution of Boneh's test (at the end of the execution of ModulusGeneration), (n, z_n) is computationally indistinguishable of (n, p_A, q_A) assuming ModulusGeneration is correct and private. A can use z_n[2] to behave (maliciously) within P. Party 1 may know the factorization of n or may have partial information regarding n which he may use in the execution. At the end of the execution of P, A knows an information set $z_{n,A,P}$ "bigger" than z_n which intuitively contains z_n and its view in P. Roughly speaking, if an adversary cannot factor an integer n (chosen at random and input by the honest party) before the execution of an nPrivate protocol, it cannot collect enough information to factor n without making the protocol P fail. Let us see informally the link between the property nPrivate and UC-security[3]. In Gamma and RSAModulusTest parties output an encryption. Under the assumption of semantic security of the underlying encryption scheme, such output cannot "help" them to factor n. In such protocols (where the output is not informative about the factorization of n), it is easy to see that UC-security implies "*nPrivateness*": otherwise the ideal case and the real case could be distinguished with non negligible probability (factorization possible in the real-case but not the ideal-case). However the converse is not true. In particular, Gamma will be shown nPrivate but trivially not UC-secure. The property nPrivate especially dedicated to our problem allows to simplify the sub-protocols Gamma and RSAModulusTest and their security proofs. The price to pay is to understand why the next generic protocol GenFact is nPrivate. It will

[2] z_n can been interpreted as *prior* information about n known by A before the execution of P (see [17]).

[3] The UC-model was developed by R. Canetti [5]. Here UC stands for universally composable, which denotes that if a protocol is UC-secure according to the formal definition, then it is secure to use in any context (where it would have been secure to use the ideal functionality).

be used to prove that Gamma and RSAModulusTest, which are restricted implementations of this generic protocol, are nPrivate.

Definition 3. *Let k be a security parameter. GenFact is a protocol between an arbitrary oracle $\mathcal{O}$ and a party 2. The oracle $\mathcal{O}$ inputs a public k-bit integer n and it is assumed that $\mathcal{O}$ answers (correctly or not, depending of the characteristics of $\mathcal{O}$) to any binary question[4] (a question whose answer is "yes" or "no"). Party 2 first asks an arbitrary binary question. If the answer is "no" the protocol fails. If the protocol does not fail, party 2 can ask a second binary question. If the answer is "no" the protocol fails and so on.... The number of questions is assumed polynomial in k.* ***Moreover, it is assumed that party 2 receives arbitrary information (e.g. the factorization of n) when the protocol fails.***

The next result states that a polynomial adversary controlling party 2 cannot "learn" the factorization of n without making GenFact fails. Intuitively, if A wants to learn r bits, GenFact fails with probability $1-2^{-r}$.

Lemma 1. *GenFact is nPrivate against any polynomial adversary A controlling party 2.*

Proof. (Sketch.) Let $D = (D_k)_{k\in\mathbb{N}}$ be an nPrivate probability distribution family (see definition 2) and A be a polynomial adversary controlling party 2. Let $(n, z_n) \leftarrow D_k$. The probability that A factors n without making GenFact fail is denoted by P_A. Let us suppose that GenFact is not nPrivate by assuming that $P_A > p(k)$ where p is a polynomial.

GenFact fails if A receives at least one "no" to any of the arbitrary binary questions mentioned in definition 3. Thus, when A chooses its i^{th} question, it has received $i-1$ positive answers[5], i.e. ("yes", "yes",...,"yes"). Thus, the received messages are not informative (the transcript of the protocol can be exactly simulated in polynomial time without knowing the factorization of n) to choose its i^{th} question. Consequently, A can choose the polynomial-size list $\mathcal{L}$ of its questions *a priori*, i.e. before the execution of GenFact.

Let us show that there exists a polynomial adversary A', only given n, z_n, able to factor n with probability larger than $p(k)$ without interacting with the oracle $\mathcal{O}$. Indeed, A' chooses the same list $\mathcal{L}$ of questions *a priori*, assumes that the answers are "yes" and then computes the factorization

[4] For instance, "Does God exist?" is a binary question. In order to factor n, the party 2 should ask questions relative to n, e.g. n is a RSA-modulus $n = pq$? Is the i^{th} bit of p equal to 1? etc...

[5] Otherwise the protocol has failed previously.

of n as A would do (one could imagine that A' invokes A on the same list $\mathcal{L}$ of questions with positive answers). We consider the three following events S, F, F':

1. $S =$ Genfact does not fail.
2. $F = A$ factors n
3. $F' = A'$ factors n

The protocol does not fail when all the questions of $\mathcal{L}$ get a positive answer. In this case, A' correctly assumes that all the answers are positive implying that A and A' have the same information: consequently they factor n with the same probability (by construction of A'), i.e.

$$P(F|S) = P(F'|S)$$

By noting that $P_A = P(F|S)P(S)$, we have

$$P(F') \geq P(F'|S)P(S) = P(F|S)P(S) = P_A$$

It implies that $P(F') \geq P_A > p(k)$ meaning that A' factors n with non negligible probability. It contradicts that D is nPrivate. Consequently, $P_A < p(k)$ proving that GenFact is nPrivate.
$\square$

As expected, this lemma implies that Boneh's test is nPrivate (but not correct) against any polynomial adversary. Indeed, this protocol can be seen as a restricted implementation of GenFact where the oracle is replaced by the honest party and the i^{th} binary question is $t_i a^{s_B} \overset{?}{\equiv} 1 \mod n$, t_i being an arbitrary value chosen by A (recall that in GenFact, party 2 receives arbitrary information if the protocol fails: in Boneh's test, it receives $t_i a^{s_B}$).

The next intuitive result (see appendix C for the proof) states that the property nPrivate is stable by (serial) composition.

Proposition 1. ***(Composition.)*** *If R and S are two nPrivate protocols then the composed protocol $P = R \to S$ consisting of executing R and S successively (where honest parties input the same integer n in R and S) is also nPrivate.*

3 RSA Moduli Generation

In this section, we propose a secure (according to the security requirements presented in the previous section) implementation of the main protocol of this paper, i.e. RSAModulusGeneration. This protocol is based on the protocols RSAModulusTest and Gamma as schematically explained in the introduction. These protocols assume the existence of a semantically secure additively homomorphic public key encryption scheme $S = (G, E, D)$. It is also assumed the existence of the following UC-secure protocols:

1. CorrectKey is a ZK-proof proving correctness of keys.
2. EncryptProof is a ZK-proof allowing the private key owner to prove that a public encryption X encrypts a public value x without revealing anything about the private key.
3. Multiplication. Alice has 2 secret encryptions $X = E_B(x)$ and $Y = E_B(y)$ of 2 unknown values x and y. She outputs an encryption of Z of xy, i.e. $Z = E_B(xy)$, without learning and without revealing anything about x and y.
4. Bound is ZK-proof allowing Alice to prove that an encryption $X_A = E_A(x)$ is a valid encryption of a value x smaller than some public threshold B without revealing anything else about x.

Paillier's encryption scheme is a good candidate. Indeed to prove that X encrypts x, the private key owner sends the random value r of the encryption, i.e. $X = g^x r^n \mod n^2$, obtained in the decryption phase[6]. A version of CorrectKey (which consists of proving that n and $\phi(n)$ are co-prime) can be found in [13]. A version of Multiplication can be found in appendix B and A version of Bound can be found in [8]. The classic solution for the protocol Bound is to provide encryptions to the individual bits and prove in zero-knowledge that they are bits using a zero-knowledge proof (e.g. see [8]). A more sophisticated solution for Bound requiring only O(1) exponentiations is presented in [13].

Throughout the paper, the security of the protocols will be studied in the (CorrectKey, EncryptProof, Bound and Multiplication)-hybrid model where these protocols can be replaced by an oracle. According to the composition theorem (see [5]), provided that CorrectKey, EncryptProof, Bound and Multiplication are UC-secure, the security in the hybrid model ensures security in the classical model.

In this paper, Alice (resp. Bob) generates an encryption function E_A (resp. E_B) defined over Z_{n_A} (resp. Z_{n_B}) where n_A (resp. n_B) is a $5k$-size integer. Generally, X_A (resp. X_B) will refer to an encryption done with E_A (resp. E_B). Parties invoke CorrectKey to prove correctness of keys.

[6] El Gamal's encryption scheme does not satisfy this property.

3.1 Protocol Gamma.

In this section, we suppose that the parties have already computed $n = (p_A + p_B)(q_A + q_B)$ with ModulusGeneration assumed correct and private: p_A, q_A (resp. p_B, q_B) refer to the secret values input by Alice (resp. Bob) in this protocol. Let us recall that p_A, q_A, p_B, q_B are expected to be k-bit integers such that $p_A \equiv q_A \equiv 3 \mod 4$ and $p_B \equiv q_B \equiv 0 \mod 4$. ModulusGeneration is not required to check the size of p_A, q_A, p_B, q_B because it is done here (and thus only once). Gamma aims to securely compute an encryption of $\gamma = (p-1)(q-1)/2$ such that $n = pq$. For concreteness, at the end of the execution of Gamma, Alice obtains an encryption $\varGamma_B = E_B(\gamma)$; Bob does not output anything ensuring correctness against any adversary controlling Alice. Let A be a polynomial adversary controlling Bob. If the parties honestly behave then $p = p_A + p_B$ and $q = q_A + q_B$. If the adversary modifies Bob's inputs, it might happen that $p \neq p_A + p_B$ or $q \neq q_A + q_B$ but it is ensured that $pq = n$ with p and q two k-bit such that $p \equiv q \equiv 3 \mod 4$. In this sense, Gamma is **correct** against any polynomial adversary Bob. We denote the secret $(k-2)$-bit size integers by p'_A, q'_A, p'_B, q'_B such that $p'_A = (p_A - 3)/4$, $q'_A = (q_A - 3)/4$, $p'_B = p_B/4$ and $q'_B = q_B/4$. Bob publicizes encryptions of p'_B and q'_B and proves that these encryptions encrypt $(k-2)$-bit integers with Bound.

By using homomorphic properties, Alice gets encryptions P_B and Q_B of p and q. Alice then invokes Multiplication to get an encryption N_B of pq and checks (by asking Bob to decrypt it) that $pq \equiv n$. She then invokes Multiplication to securely compute an encryption of $\gamma = (p-1)(q-1)/2$.

Proposition 2. *Assume that the encryption scheme S is semantically secure. Gamma is correct and nPrivate against any polynomial adversary in the (Multiplication, Bound, EncryptProof)-hybrid model.*

Proof. The protocol is intrinsically correct against adversary controlling Alice because Bob is not expected to output anything. Let A be a polynomial adversary controlling Bob. In the (Multiplication, Bound, EncryptProof)-hybrid model, A does not interact with Alice except in the choice of the input values. Let us imagine that A inputs $(k-2)$-bit integers p''_B and q''_B such that $p''_B \neq p'_B$ or $q''_B \neq q'_B$. In this case, either the protocol fails (step 3) or $\varGamma_B$ encrypts $(p'-1)(q'-1)2^{-1} \mod n_B$ with $p' = 4(p'_A + p''_B) + 3$, $q' = 4(q'_A + q''_B) + 3$ and $p'q' \equiv n \mod n_B$ (recall that E_B is an encryption function over Z_{n_B}). As n_B was chosen sufficiently large ($5k$-bit integer), there are not modular reductions and the equalities remain true over Z).

Protocol 1 : Gamma.

Require : Let k be a security parameter. Let n be a public composite odd integer. Alice (resp. Bob) knows two $(k-2)$-bit integers p'_A, q'_A (resp. p'_B, q'_B) such that $n = (4(p'_A + p'_B) + 3)(4(q'_A + q'_B) + 3)$.

1. **Bob** publicizes encryptions $A = E_B(p'_B)$ and $B = E_B(q'_B)$. He then proves that A and B encrypt a $(k-2)$-bit integer by invoking Bound.
2. **Alice** computes, by using homomorphic properties, encryptions P_B, Q_B of respectively $p = 4(p'_B + p'_A) + 3$ and $q = 4(q'_B + q'_A) + 3$. By invoking Multiplication, she then computes an encryption N_B of pq.
3. **Alice** sends N_B and **Bob** decrypts it and he sends the decrypted value and proves the decryption with EncryptProof. If N_B does not encrypt n, the protocol **fails**.
4. **Alice** computes encryptions of $p-1$ and $q-1$ by using homomorphic properties and invokes Multiplication on these encryptions to compute an encryption of $(p-1)(q-1)$. By using homomorphic properties She computes and **outputs** an encryption Γ_B of $(p-1)(q-1)/2$, i.e. $2^{-1}(p-1)(q-1) \mod n_B$.

In the (Multiplication, Bound, EncryptProof)-hybrid model, the protocols Multiplication, Bound, EncryptProof can be replaced by a trusted party. Thus, the steps (1)+(2) and the step (4) can be seen as sub-protocols without direct interaction between parties (just with the trusted party). Moreover parties only receive encryptions (and "true" from the trusted party simulating Bound). Assuming S semantically secure, these sub-protocols are nPrivate. According to the composition property (see proposition 1), it suffices to prove that step (3) (interpreted as a sub-protocol) is nPrivate.

Let A be a polynomial adversary controlling Alice. In step 3, A can send any encryption N_B. However if N_B does not encrypt n, the protocol fails. This step can be seen as a restricted implementation of the nPrivate protocol GenFact, described in section 2, where A asks only one question, i.e. $n =^? D_B(N_B)$. It proves that Gamma is nPrivate against any polynomial adversary controlling Alice.

Let A be a polynomial adversary controlling Bob. In the same way, A can send encryptions of p''_B and q''_B such that $p''_B \neq p'_B$ or $q''_B \neq q'_B$. The protocol fails if N_B does not encrypt n. Thus Gamma can be also seen for A as a restricted implementation of GenFact, described in section 2, where A only asks if $n =^? (4(p_A + p''_B) + 3)(4(q_A + q''_B) + 3)$. According to lemma 1, Gamma is nPrivate against any polynomial adversary controlling Bob.

□

It is easy to see that that Gamma is not UC-secure. Indeed, A can send an encryption N_B of p_B for instance. By doing this, A can learn the Bob's private value p_B but the protocol would fail in step 3. To avoid this, it suffices that Bob checks that $D_B(N_B) = n$ before to send $D_B(N_B)$.

3.2 Protocol RSAModulusTest

RSAModulusTest is used to check that a non-RSA modulus has not erroneously succeeded Boneh's test (because of a malicious behavior). Thus, RSAModulusTest simply consists of checking that n is an RSA modulus. The party 1, Alice by convention, wants to be convinced with a high probability that n is an RSA modulus: Alice (note that Bob does not output anything) outputs ok if n is an RSA modulus and otherwise RSAModulusTest fails (in RSAModulusGeneration, at the beginning of the execution of RSAModulusTest, n should be an RSA modulus if parties honestly behaved). RSAModulusTest implements the following test derived from the classical Boneh's test:

1. choose at random $a \in Z_n^*$ such that $(\frac{a}{n}) = 1$ and choose a random integer $\alpha \in \{n, n+1\}$
2. compute $v = a^{\alpha\gamma} \bmod n$ with $\gamma = (p-1)(q-1)/2$
3. If $v \neq 1$ then n is not an RSA modulus.

This test is less efficient than Boneh's test in the sense that a non-RSA modulus n has a larger probability of succeeding the test. Indeed, if $a^\gamma \neq 1 \bmod n$, the probability (over the choice of α) that $a^{\alpha\gamma/2} = 1 \bmod n$ is less than $1/2$ (because $a^{(n+1)\gamma/2} = a^{n\gamma}a^\gamma$). As the probability (over the choice of $a \in Z_n$ assuming that p, q are not Carmichael integers) that $a^\gamma \neq 1 \bmod n$ is larger than $1/2$ (see [4]), the probability (over the independent choices of α and a) that $a^{\alpha\gamma} \equiv 1 \bmod n$ is smaller than $3/4$.

In RSAModulusTest, $\alpha\gamma$ is shared by using homomorphic properties of the underlying homomorphic cryptosystem S. For concreteness, Alice has a secret encryption Γ_B of γ with $pq = n$. Alice and Bob jointly choose a random number a over Z_n^* such that $(\frac{a}{n}) = 1$. Then, she randomly chooses $\alpha \in \{n, n+1\}$, randomly chooses $s_1 \in [\frac{1}{8}\alpha n, \frac{3}{8}\alpha n] \bigcap \mathbb{N}$, computes $t_1 = a^{s_1} \bmod n$, commits it and sends an encryption of $s_2 = \alpha\gamma - s_1$.[7] Bob decrypts it and sends $t_2 = a^{s_2} \bmod n$. If $t_2 t_1 \neq 1 \bmod n$ the test fails. Let us suppose that n is not an RSA modulus and that an adversary A controlling Bob has guessed $\gamma \neq \lambda(n)/2$ before the execution

[7] Note that these numbers can be encrypted by E_B defined over Z_{n_B} because n_B is assumed to be large enough, i.e. a $5k$-bit integer.

of RSAModulusTest. As s_2 has been randomly chosen in a set statistically indistinguishable from $[\frac{1}{8}n^2, \frac{3}{8}n^2] \bigcap \mathbb{N}$, A is shown unable to choose $t_2 \in Z_n^*$, with probability larger than 3/4, such that $t_1 t_2 = 1 \bmod n$. This ensures correctness against any polynomial adversary controlling Bob. RSAModulusTest is trivially nPrivate against any polynomial adversary. Furthermore, RSAModulusTest can be seen, for Alice, as an instance of the generic protocol GenFact. It ensures that this protocol is nPrivate against any polynomial adversary controlling Alice (according to lemma 1).

Protocol 2 : RSAModulusTest

Require : Let k be a security parameter and $l \in \mathbb{N}$ be indexed by k. Let $n = pq$ be an odd composite integer where p and q are k-bit integers such that $p \equiv q \equiv 3 \bmod 4$. Party 1 (Alice by convention) has a secret encryption $\mathit{\Gamma}_B$ of $\gamma = (p-1)(q-1)/2$.

For $i = 1$ to l

1. **Alice and Bob** jointly choose $a \in Z_n^*$ at random such that $(\frac{a}{n}) = 1$
2. **Alice** randomly chooses $\alpha \in \{n, n+1\}$ and $s_1 \in [\frac{1}{8}\alpha n, \frac{3}{8}\alpha n] \bigcap \mathbb{N}$ and sends $U = E_B(\alpha\gamma - s_1)$ where U is computed only by using homomorphic properties. She computes an encryption T_1 of $t_1 = a^{s_1} \bmod n$ with the encryption function E_A, i.e. $T_1 = E_A(t_1)$ and sends it (t_1 is committed).
3. **Bob** decrypts U, i.e $s_2 = D_B(U)$ and sends $t_2 = a^{s_2} \bmod n$
4. **Alice** sends $t_1 = D_A(T_1)$ and proves the decryption with EncryptProof.
5. **Alice and Bob** compute $v = t_1 t_2 \bmod n$. If $v \neq 1$ then the protocol **fails**.

End for

Alice outputs ok

Proposition 3. *Assuming the encryption scheme S is semantically secure, RSAModulusTest is correct and nPrivate against any polynomial adversary in the (EncryptProof)-hybrid model.*

Proof. First, let us note that the encryption domain Z_{n_B} of E_B was chosen large enough (n_B is a $5k$-bit integer and all the considered integers are at most $4k$-bit integers) to avoid modular reductions[8] (modulo n_B). Let A be a polynomial adversary controlling Bob. Let us prove that RSAModulusTest is **correct** against A. As the only possible output is ok, RSAModulusTest is intrinsically correct if n is an RSA modulus. Let us suppose

[8] It can be imagined that the encryption domain of E_B is $\mathbb{N}$.

that n is **not** an RSA modulus such that A knows its factorization, $\lambda(n)$ and $\gamma \neq \lambda(n)/2$. RSAModulusTest is correct if Alice outputs ok with a negligible probability. Let [9] $a \in Z_n^*$ s.t. $a^\gamma \neq 1 \bmod n$. In order to make Alice output ok, A should guess, at each iteration, $t_1^{-1} \mod n$ (and send it to Alice at step 3). If A knows γ, it knows that

$$t_1^{-1} \in \{a^{s_2 - i\gamma} \mod n | i \in \{n, n+1\}\}$$

By assuming $a^\gamma \neq 1 \mod n$, this set contains two distinct elements m_1 and m_2. A receives a value s_2 such that $s_1 + s_2 = \alpha\gamma$ (the equality holds because the domain size Z_{n_B} of E_B is sufficiently large) where s_2 is a random number over a distribution statistically indistinguishable to the uniform distribution over $[\frac{1}{8}n^2, \frac{3}{8}n^2] \bigcap \mathbb{N}$: the statistical indistinguishability holds by noticing that $\gamma - n/2 = O(n^{\frac{1}{2}})$ (here, it is assumed that p and q are k-bit integers) is negligible implying that for any $\alpha \in \{n, n+1\}$

$$[\alpha\gamma - \frac{3}{8}\alpha n, \alpha\gamma - \frac{1}{8}\alpha n] \bigcap \mathbb{N} \overset{s}{\equiv} [\frac{1}{8}n^2, \frac{3}{8}n^2] \bigcap \mathbb{N}$$

Consequently, A cannot distinguish between $\alpha = n$ or $\alpha = n+1$ implying that it cannot distinguish between $t_1^{-1} \equiv m_1 \mod n$ or $t_1^{-1} \equiv m_2 \mod n$ with probability significantly larger than 1/2. The probability to choose $a \in Z_n^*$ such that $a^\gamma \neq 1 \bmod n$ is larger than 1/2 see [4][10]. Therefore the probability that A guess $t_1^{-1} \mod n$ is smaller than 3/4. Also, the probability to do this at each iteration is smaller than $(3/4)^k$.

RSAModulusTest is trivially nPrivate against A. Indeed, the view of A (when the protocol does not fail) can be simulated by a list of k independent triplets of values (T_1, a^{-s_2}, s_2) where T_1 is randomly chosen in the ciphertext domain of E_A, s_2 is chosen at random in a set statistically indistinguishable from $[\frac{1}{8}n^2, \frac{3}{8}n^2] \bigcap \mathbb{N}$. Assuming the semantic security of S, the simulation and the real view are computationally indistinguishable.

Let A be a polynomial adversary controlling Alice. Correctness is implicit because Bob does not output anything. RSAModulusTest can be seen, for A, as a restricted implementation of GenFact, described in section 2, where the k questions are $a^{s_2} \equiv^? t_1^{-1} \mod n$ (the commitment of t_1 is required to get this). According to lemma 1, RSAModulusTest is nPrivate against any polynomial adversary controlling Alice.

□

[9] Throughout this paper, we neglect the probability that n is a Carmichael modulus.
[10] By supposing that n is not a Carmichael RSA modulus.

3.3 Protocol RSAModulusGeneration

The following protocol RSAModulusGeneration implements the functionality described in section 2. As mentioned in section 2, ModulusGeneration denotes a correct and private (in the malicious case) protocol computing $n = (p_A + p_B)(q_A + q_B)$. This protocol is used to build an RSA modulus candidate $n = pq$. The parties then use the classical Boneh's test to test if n is an RSA modulus. If not, the parties exchange the current values p_A, p_A, p_B, q_B (which do not need to be kept secret anymore) and check that the other party behaved honestly before generating an other modulus n (by executing a new iteration of the loop). If Boneh's test claims that n is an RSA modulus, the parties then invoke Gamma to get an encryption of $(p - 1)(q - 1)/2$. They then check that n is really an RSA modulus by executing RSAModulusTest twice (exchanging their positions each time). The idea is to execute RSAModulusTest only if n is an RSA modulus or if the adversary did not behave honestly. RSAModulusTest will distinguish these two cases. Note that either RSAModulusGeneration outputs the first RSA modulus n_0 output by ModulusGeneration in step 1.a.: it will be used in the security analysis of RSAModulusGeneration.

Theorem 1. *Let A be a polynomial adversary controlling Alice or Bob. Assume that the encryption scheme S is semantically secure, the following assertions are true in the (CorrectKey, EncryptProof, Bound and Multiplication)-hybrid model.*

1. *RSAModulusGeneration is correct against A, i.e. the probability that the output n is not an RSA modulus is negligible.*
2. *Assuming conjecture 1, the probability that the protocol does not fail and that A factors the output n is negligible.*

Proof. Let k be the security parameter. Let us suppose that Bob is honest and that Alice is controlled by an active adversary A.

ModulusGeneration is assumed to correctly compute $n = (p_A+p_B)(q_A+q_B)$. If n is not an RSA modulus and A cheats Boneh's test in step (1.b), the protocol fails in step (4) according to proposition 1. Thus, RSAModulusGeneration cannot output a non-RSA Modulus. If p_A or q_A is not a k-bit size integer, if $p_A \not\equiv 3 \mod 4$ or if $q_A \not\equiv 3 \mod 4$ then the protocol fails in step 2. Thus, the output is a well-formed RSA modulus. It proves correctness.

Let us show that either the first RSA modulus generated by ModulusGeneration is output or RSAModulusGeneration fails. Let $n_0 = pq = (p_A + p_B)(q_A + q_B)$ be the **first** (by assuming the protocol has not failed

Protocol 3 : RSAModulusGeneration

Require : Let k be a security parameter. Alice (resp. Bob) generates an encryption function E_A (resp. E_B) defined over Z_{n_A} (resp. Z_{n_B}) where n_A (resp. n_B) is a $5k$-size integer. Parties invoke CorrectKey to prove correctness of keys.

1. *While Boneh's test fails*
   (a) **Alice and Bob** randomly choose k-bit numbers, respectively p_A, q_A and p_B, q_B such that $p_A \equiv q_A \equiv 3 \mod 4$ and $p_B \equiv q_B \equiv 0 \mod 4$. They then invoke ModulusGeneration to get $n = (p_A + p_B)(q_A + q_B)$.
   (b) **Alice and Bob** invoke Boneh's test to decide if n is an RSA modulus or not.
   (c) if n failed Boneh's test then parties broadcast p_A, q_A, p_B, q_B and verify that $n = (p_A + p_B)(q_A + q_B)$ and $p_A + p_B$ or $q_A + q_B$ are not prime. If n is an RSA modulus then the protocol fails.

   End while
2. **Alice and Bob** invoke Gamma twice (by exchanging their position) on the private inputs $((p_A - 3)/4, (q_A - 3)/4)$, $(p_B/4, q_B/4)$ and the public input n. Alice gets the encryption Γ_B and Bob gets the encryption Γ_A.
3. **Alice** invokes RSAModulusTest on the input n, Γ_B (Alice is party 1)
4. **Bob** invokes RSAModulusTest on the input n, Γ_A (Bob is party 1)
5. If **Alice and Bob** have output ok in the previous steps, n **is output.**

before) RSA modulus output by ModulusGeneration in loop 1. If A cheats Boneh's test in step (1.b) then the protocol fails in step (1.c) else the step 2 is executed. If p or q are not k-bit size integers then the execution of Gamma makes RSAModulusGeneration fails in step 2. As RSAModulusTest is correct, either n_0 is output or the protocol fails in step 4. Thus, the output RSA modulus n_0 is randomly chosen such that $p \equiv q \equiv 3 \mod 4$, $p \in [p_A + 2^k, p_A + 2^{k+1}[\bigcap \mathbb{N}$ and $q \in [q_A + 2^k, q_A + 2^{k+1}[\bigcap \mathbb{N}$ where p_A, q_A are two arbitrary k-bit integers (chosen by the adversary). According to conjecture 1 and assuming that ModulusGeneration is assumed to be private, A cannot factor n_0, with non negligible probability, at the end of step 1.a.. Indeed, in the ideal case (see [5]) an adversary A_I knows n, p_A, q_A and in the real case the adversary A knows $n_0, z_{n_0,A}$ ($z_{n_0,A}$ being the view of A in ModulusGeneration). Indeed, according to conjecture 1, there there does not exists any polynomial adversary A_I able to factor n_0 with non negligible probability in the ideal case (given n_0, p_A, q_A). Thus, if there exists a polynomial adversary A able to factor n_0 with non negligible probability, then the real view and the ideal view can be distinguished with non negligible probability: it contradicts that ModulusGeneration is private. Thus such an adversary A does not exist. In other words, by denoting D_k the distribution of $n_0, z_{n_0,A}$, the family of distributions $(D_k)_{k\in\mathbb{N}}$ is nPrivate. Boneh's test is nPrivate and according to proposition 2 and 3, Gamma and RSAModulusTest are nPrivate. According to the composition property (see proposition 1), the sub-protocols composed by steps (1b.)(2),(3) and (4) is nPrivate. According to the definition 2, A cannot output the factorization of n_0 without making the nPrivate sub-protocol composed of the steps (1.b.)(2),(3),(4) (thus RSAModulusGeneration) fail with non negligible probability.

$\square$

Efficiency Analysis. Classical Protocols generating RSA moduli correspond to the steps (1a) (1b) of RSAModulusGeneration. The steps (1c) (2), (3), (4) were introduced in order to make the protocol robust against active adversaries. Let us evaluate the supplementary cost of these steps.

In a first analysis, we only take into account modular exponentiations, encryptions and decryptions (ME/E/D) that Alice (and symmetrically Bob) must compute (neglecting, for instance, modular multiplications). The underlying encryption scheme S is assumed to be the Paillier cryptosystem. The number of iterations of the loop is approximatively equal to $\log^2 n$ on average. As (1c) consists of doing $\log^2 n$ primary tests, the cost of (1c) is approximately equal to $\log^2 n$ exponentiations. However, (1c) should be added in all classical Boneh's test protocols to not allow an adversary to cheat Boneh's test on RSA moduli. Furthermore, (1c) does not need to be executed at each iteration but only with a given probability.

Gamma and RSAModulusTest are executed twice. Gamma requires $O(\log n)$ ME/E/D (assuming Bound requires $O(1)$ ME/E/D) and RSAModulusTest requires $O(1)$ ME/E/D. Thus, steps (2),(3),(4) can be neglected with respect to step (1) (which requires $O(\log^2 n)$ ME/E/D) when $\log n$ is large. Let us detail our analysis.

In Gamma, the protocol Bound is invoked twice. For classical security level, approximatively 10^3 ME/E/D are required. Without any optimization, the total running time of steps (2), (3), (4) is a few minute on a standard platform.

However, as far as we know, there does not exist two-party really efficient Boneh's test based protocols (while there exists such efficient protocol in the multi-party case [16]): hours are required to output a 1024-bit RSA modulus. It is very challenging to find such a protocol.

Complementary approaches to reach efficiency consist of modifying the distribution of the inputs (see [15]). For instance, it should be asked to Alice to choose p_A, q_A as multiples of some small primes and it should be asked to Bob to choose p_B, q_B as multiples of **other** small primes. By doing this $p_A + p_B$ and $q_A + q_B$ are not multiple of the involved small primes increasing their probability to be prime.

4 Conclusion and future work

We first showed that existing two-party protocols based on Boneh's test are not secure against actives adversaries. In this paper, we have provided a provably secure (with security requirements dedicated to our application) protocol to solve this problem. We propose a simple and efficient "patch" making most Boneh's test-based protocols correct against any polynomial active adversaries. We develop *ad-hoc* security tools to prove that a polynomial adversary cannot learn the factorization of n without making the protocol fail. These new security tools could be re-used for other applications. A natural extension would consist of considering the multi-party case in presence of an arbitrary number of misbehaving parties.

Acknowledgements. The authors thank the reviewers for their helpful comments.

References

1. J. Algesheimer, J. Camenisch, and V. Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *CRYPTO*, pages 417–432, 2002.
2. Simon R. Blackburn, Simon Blake-Wilson, Mike Burmester, and Steven D. Galbraith. Weaknesses in shared rsa key generation protocols. In *IMA Int. Conf.*, pages 300–306, 1999.
3. D. Boneh and M.K. Franklin. Efficient generation of shared rsa keys (extended abstract). In *CRYPTO*, pages 425–439, 1997.
4. D. Boneh and M.K. Franklin. Efficient generation of shared rsa keys. *J. ACM*, 48(4):702–722, 2001.
5. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
6. R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001.
7. I. Damgård and K. Dupont. Efficient threshold rsa signatures with general moduli and no extra assumptions. In *Public Key Cryptography*, pages 346–361, 2005.
8. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136, 2001.
9. Ivan Damgård and Maciej Koprowski. Practical threshold rsa signatures without a trusted dealer. In *EUROCRYPT*, pages 152–165, 2001.
10. Ivan Damgård and Gert Læssøe Mikkelsen. Efficient, robust and constant-round distributed rsa key generation. In *TCC*, pages 183–200, 2010.
11. P. Fouque and J. Stern. Fully distributed threshold rsa under standard assumptions. In *IACR Cryptology ePrint Archive: Report 2001/2008*, February 2001.
12. N. Gilboa. Two party rsa key generation. In *CRYPTO*, pages 116–129, 1999.
13. Carmit Hazay, Gert Lsse Mikkelsen, Tal Rabin, and Tomas Toft. Efficient rsa key generation and threshold paillier in the two-party setting. Cryptology ePrint Archive, Report 2011/494, 2011. http://eprint.iacr.org/.
14. Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *CRYPTO*, pages 36–54, 2000.
15. Daniel Loebenberger and Michael Nüsken. Analyzing standards for rsa integers. In *AFRICACRYPT*, pages 260–277, 2011.
16. Michael Malkin, Thomas D. Wu, and Dan Boneh. Experimenting with shared generation of rsa keys. In *NDSS*, 1999.
17. O.Goldreich, S.Michali, and A.Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
18. G. Poupard and J. Stern. Generation of shared rsa keys by two parties. In *Asiacrypt'98, LNCS 1514, Springer-Verlag*, pages 11–24, 1998.
19. Tal Rabin. A simplified approach to threshold and proactive rsa. In *CRYPTO*, pages 89–104, 1998.
20. Victor Shoup. Practical threshold signatures. In *EUROCRYPT*, pages 207–220, 2000.
21. T. Straub. Efficient two party multi-prime rsa key generation. In *IASTED International Conference on Communication, Network, and Information Security*, 2003.
22. P. MacKenzie Y. Frankel and M. Yung. Robust efficient distributed rsa key generation. In *STOC'98*, pages 663–672, 1998.

A Attack on Boneh's test

Boneh and Franklin [3] have proposed a test to decide if $n = pq$ is an RSA modulus assuming that $p \equiv q \equiv 3 \mod 4$. This test consists of randomly choosing $a \in Z_n^*$ such that $(\frac{a}{n}) = 1$, computing[11] $v = a^{(p-1)(q-1)/2} \mod n$ and checking if $v \neq 1$. For a value n which passes the test t times, n is not an RSA modulus with probability smaller than 2^{-t}. This test is not a complete probabilistic test in the sense that there are some integers n, which are not RSA moduli, but which succeed the test for any basis a. However, this probability can be considered as negligible (around 10^{-60}) for the number sizes considered in our application. In existing protocols [4], [12], [18], [21], Alice and Bob generate $n = (p_A + p_B)(q_A + q_B)$ with a secure protocol ModulusGeneration where $p_A \equiv q_A \equiv 3 \mod 4$ and $p_B \equiv q_B \equiv 0 \mod 4$. Alice then computes $s_A = (n - p_A - q_A + 1)/2$ and Bob computes $s_B = -(p_B + q_B)/2$ in order to additively share $(p-1)(q-1)/2$, i.e. $s_A + s_B = (p-1)(q-1)/2$. Finally, Alice computes $a^{s_A} \mod n$, Bob computes $a^{s_B} \mod n$ and they check that $a^{s_A}a^{s_B} = 1 \mod n$. This test is secure against passive adversary. However, this protocol may be insecure against an active adversary because there is nothing to ensure that parties send the correct value $a^s \mod n$ in Boneh's test. To secure the protocol, each party should prove that it sends $a^s \mod n$. Classical methods (based on zero-knowledge proofs which ensure two values $a^s \mod n$ and $g^s \mod n$ have the same discrete logarithm s) cannot be used because the shared values s are chosen before n (they are used to build n). In the next section, an attack in a dissymmetric environment is presented and evaluated (note that nothing excludes that other attacks exist, even in a symmetric environment).

A.1 Presentation of a brute force attack

This attack works in a dissymmetric environment where one party has more resources than the other.[12] Let us suppose that an active polynomial adversary A controls Alice. We suppose that A has more resources than Bob. We denote the ratio of resources between Alice and Bob by $r > 1$.

[11] The original Boneh test consists of checking if $a^{(p-1)(q-1)/4} = \pm 1 \mod n$. This formulation is equivalent to ours. Indeed, as it is assumed that $p \equiv q \equiv 3 \mod 4$, 1 and -1 are the two square roots of 1 with a Jacobi symbol equal to 1. Thus, $a^{(p-1)(q-1)/2} = 1 \Leftrightarrow a^{(p-1)(q-1)/4} = \pm 1$.

[12] The "power " party ressource can be, for instance, only linear with respect to the resource of the other. It is obviously assumed that the "power" party is not able to factor output RSA moduli, i.e. 1024-bit RSA moduli.

Typically, A computes r times faster than Bob. In Boneh's test, each party computes a modular exponentiation. We denote the time needed by Bob to compute the modular exponentiation, by t_e. A can try to factorize n during $(1 - 1/r)t_e$ time units and keeps t_e/r time units to compute a modular exponentiation. Let us suppose that $q = q_A + q_B$ is prime and that Alice manages to decompose n in prime factors, i.e. $n = p_1 p_2 ... p_m q$. This situation can be identified by A with probability $1/2$ (A cannot distinguish it from the symmetric case where $p = p_A + p_B$ is prime) because $|q| = k > |p_i|$. Thus, A can deduce $p_B = p_1 p_2 ... p_m - p_A$ and $q_B = q - q_A$. Consequently A can compute Bob's secret share s_B. By sending $a^{-s_B} \mod n$ instead of $a^{s_A} \mod n$, Boneh's test succeeds and Bob is convinced that n is an RSA modulus. A first naive approach would consist of letting time t_B to Bob to test if n is a RSA modulus after the Boneh's test. In the next section, we experiment with this type of attack against this naive approach.

A.2 Concrete scenario of the attack

In this section, Bob's simulations were implemented on a standard PC (Pentium 2 GHz) with the java class *BigInteger*. The factorization algorithm is ECM. This algorithm is particularly efficient for discovering *small* factors in large integers. In a sense this algorithm is very relevant for Bob's problem which consists of checking whether n is an RSA modulus or not. In our simulations, A computes on the same platform but the obtained computations times are divided r.

In order to correctly evaluate this attack, we should evaluate the probability of getting "malicious" values of n. A malicious $n = p_1 ... p_m q$ can be factored in a small amount of time by A while Bob cannot find any factor p_i. Typically, $p_1, .., p_m$ should be large enough but not too large. In the following experiment, we simulate the attack with the following parameters $|n| = 1024$, $r = 10^5$ and $t_e = 100ms$ (this is approximately the time that Bob needs to compute a modular exponentiation considering 1024-bit size integers). We denote the time available to Bob after applying Boneh's test in order to check if n is an RSA modulus by t_B. We wish to estimate the probability of finding n such that A is able to factorize it in $(1 - 1/r)t_e \approx t_e$ time units and such that Bob is not able to find a factor within t_B time units. In our experiments, if $|p_i| \in \{80, ..., 110\}$ (and $|q| = 512$) then A is able to factorize n within $t_e = 100ms$ (3h with our platform) and Bob needs at least 3mins to find a factor.

Now, let us evaluate the ratio between the probability of getting an RSA modulus and the probability of finding such malicious values of n.

This ratio provides a lower bound of the probability to output a non-RSA modulus. A short analysis [13] based on prime theorem shows that this ratio is approximately equal to 10^{-4}.

To summarize, if the ratio of performance between Bob and A is larger than 10^5 and if Bob is given less than 3 minutes after applying Boneh's test in order to verify whether n is an RSA modulus, then the adversary is able to convince Bob that a non-RSA modulus is a RSA-modulus with probability larger than 10^{-4}.

In this paper, we propose a *"patch"* to the classical Boneh's test in order to ensure that a value n is an RSA modulus against any polynomial adversary, in particular against a more powerful adversary (whatever the performance ratio r is). The computation of this patch requires only few seconds.

B Protocol Multiplication

MultProof. A party builds 3 encryptions X, Y, Z from 3 values x, y, z such that $z = xy$. MultProof is a $\Sigma-$ protocol which proves that Z encrypts xy without revealing anything about x and y.

A version of MultProof is dedicated to Paillier's encryption scheme. It can be found in [6] (generic versions can also be found). We have as inputs encryption $C_x = g^x r^n \mod n^2$, $C_y = g^y s^n \mod n^2$, $D = C_x^y \gamma^n \mod n^2$ and a player P_i who knows in addition s, y, γ. What we need is a proof that D encrypts $xy \mod n$. We proceed as follows:

1. P_i chooses $a \in Z_n$ and $b, c \in Z_n^*$ at random, computes and sends
$$A = C_x^a b^n \mod n^2, B = g^a c^n \mod n^2$$
2. The verifier sends a random challenge e
3. P_i computes and sends
$$w = a + ey \mod n, z = cs^e g^t \mod n^2, z' = bC_x^t \gamma^e \mod n^2$$
where t is defined by $a + ey = w + tn$.

[13] We assume that the number of primes smaller than any integer t is equal to $t/\log t$. By using this approximation, we can estimate the number P_k of primes of size k. To estimate the number of 512-bit size integers $p = p_1...p_m$ with $|p_i| \in \{80, ..., 110\}$, it suffices to consider each possible value of $7 \geq m \geq 5$, each possible size (without taking into account permutations) of $p_1, p_2, ..., p_m$ belonging to $\{80, ..., 110\}$. Given m and $|p_1|, ..., |p_m|$, it is easy to count the numbers of 512-bit size integers p knowing $P_{|p_1|}, ..., P_{|p_m|}$. By considering all size configurations, we approximately obtain the number of 512-bit size integers which can be written as product of $\{80, ..., 110\}$-bit size primes p_i.

4. The verifier checks that

$$g^w z^n = BC_y^e \mod n^2, C_x^w y^n = AD^e \mod n^2$$

and accepts if and only if it is the case.

This zero-knowledge proof is shown secure under composition. Based on such proofs, a secure protocol Multiplication can be easily built.

Multiplication

Require : Alice has two encryptions $E_B(x), E_B(y)$ of unknown values $x, y \in Z_{n_B}$

1. **Alice** sends $E_B(r + x), E_B(s + y)$ where r and s are random numbers chosen in Z_{n_B}
2. **Bob** computes and sends $Z = E_B((x+r)(y+s))$. He then proves that Z encrypts the correct value with MultProof.
3. **Alice** computes an encryption of $(x+r)(y+s)-sx-ry-rs$ by using homomorphic properties and outputs it.

C Proof of proposition 1

Let $D = (D_i)_{i\in\mathbb{N}}$ be an nPrivate probability distribution family (see definition 1) and k be a security parameter. Let A be a polynomial adversary and $(n, z_n) \leftarrow D_k$. Without loss of generality, an adversary A in P can be decomposed in two adversaries A_R, A_S, denoted by $A = A_R \rightarrow A_S$ as follows: A_R receives $(n, z_n) \leftarrow D_k$ before the execution of R, outputs $(n, z_{n,A,R})$ and sends it to A_S which starts the protocol S given $(n, z_{n,A,R})$.

Let us change the rules of interaction with the adversary A. If R fails, A is given "another chance" by letting it execute S but by deleting the information it learns about n. More formally, this can be interpreted as the interaction between party 1 and a new polynomial adversary $A' = A'_R \rightarrow A_S$ in a new composed protocol $P' = R' \rightarrow S$: R' is the same protocol as R except that the A'_R outputs z_n (the honest party outputs an arbitrary value) when R fails and A'_R outputs $z_{n,A,R}$ otherwise (note that R' never fails). By summary

$$z_{n,A',R'} = \begin{cases} z_{n,A,R}, & \text{if } R \text{ does not fail;} \\ z_n, & \text{otherwise.} \end{cases}$$

It is clear that the probability that A' outputs the factorization of n without making P' fail is larger than the probability that A outputs the factorization of n without making P fail. Let us show that this probability is negligible. Let D'_k be the probability distribution of $(n, z_{n,A',R'})$. As S is $\mathsf{nPrivate}$, it suffices to prove that $(D'_k)_{k\in\mathbb{N}}$ is $\mathsf{nPrivate}$.

Let us suppose that it is not the case and that there exists a p.p.t algorithm F able to factor n, with non negligible probability, given $n, z_{n,A',R'}$. This implies that one of the two following issues is satisfied with non negligible probability:

1. F factors n given n, z_n
2. F factors n given $n, z_{n,A,R}$ and R does not fail.

If F cannot factor n, with non negligible probability, given n, z_n because $(D_k)_{k\in\mathbb{N}}$ is $\mathsf{nPrivate}$. Let A'' be a polynomial adversary behaving like A in R and applying F on $n, z_{n,A,R}$. Issue 2 implies that the probability that A'' factors n without making R fail is not negligible. Thus, issue 2 cannot hold assuming that R is $\mathsf{nPrivate}$. It proves that $(D'_k)_{k\in\mathbb{N}}$ is $\mathsf{nPrivate}$.

Enhanced Flexibility for Homomorphic Encryption Schemes via CRT

Yin Hu, William J. Martin, Berk Sunar

Worcester Polytechnic Institute

Abstract. The Chinese Remainder Theorem (CRT) has numerous applications including in cryptography. In a striking example of this utility, we demonstrate how the CRT facilitates making one additive homomorphic encryption scheme viable and making another more flexible. First we show that the CRT may be used to turn an intractable problem into a tractable one. Specifically, using the CRT to replace a single group element by a logarithmic number of elements in the same group, we lay the foundation for additively homomorphic encryption schemes using well-known and previously deployed primitives. Our solution is shown to be secure and quite general in nature. We present a simple technique for ElGamal-type encryption schemes which facilitates encryption in an additively homomorphic manner. Secondly we apply the CRT to a previous encryption scheme proposed by Boneh, Goh and Nissim that supports efficient homomorphic evaluations of 2-DNF circuits [4]. One drawback mentioned in [4] was a restriction on the size of the output message space – prompting an open problem posed by the authors. Again employing the CRT, we devise an elegant modification in which we solve the problem, supporting arbitrary output sizes.

Keywords: Homomorphic Encryption, ElGamal, CRT.

1 Introduction

One of the most significant developments in cryptography in the last few years has been the introduction of the first fully homomorphic encryption scheme by Gentry [1]. Since addition and multiplication on any non-trivial ring constitute a Turing-complete set of gates, a fully homomorphic encryption scheme – if made efficient – allows one to employ untrusted computing resources without risk of revealing sensitive data. Computation is carried out directly on ciphertexts and the true result of circuit evaluation is not revealed until the decryption stage. In addition to this powerful applicability, Gentry's lattice-based scheme appears to be secure and hence settles an open problem posed by Rivest et al. in 1978 [2].

While fully homomorphic encryption has great potential and implementations are rapidly improving, a number of important applications call for schemes with milder homomorphic properties. Such technologies as blinded data aggregation, electronic voting, biometrics, and private information retrieval may all make use of existing partially homomorphic encryption systems as soon as efficient implementations become available. Systems such as Paillier's additively homomorphic scheme [3] and the 2-DNF evaluation scheme of Boneh, et al. [4] admit implementations that are far more efficient than current implementations of the fully homomorphic Gentry scheme. In fact, it is clear that, even when fully homomorphic schemes enter the realm of practical implementation, streamlined partially homomorphic schemes tailored to particular applications of economic importance will remain worthy of study and optimization. And the study of such schemes holds theoretical value as well: the contrast between efficient candidates and more powerful ones allows us to explore new alternatives, new algebraic contexts, and new optimizations, for prospective fully homomorphic schemes. Our contribution therefore has not only immediate practical applications, but in its simplicity and generality is of theoretical interest as well.

Our Contribution. We have two contributions: first, we present a modification to the ElGamal scheme which makes additive homomorphic evaluation viable. Second, we eliminate the restriction on the circuit output size in the 2-DNF scheme introduced by Boneh, Goh and Nissim [4]. Both ideas employ the Chinese Remainder Theorem (CRT), but to slightly different effect.

It is well known that ElGamal-type encryption schemes are homomorphic with respect to one algebraic operation. However, this feature has been widely dismissed as useless since in the additive context, message recovery involves solving a discrete logarithm problem in the group, and this is precisely the problem whose difficulty ensures security. Our solution is simple: we employ the CRT to replace one discrete logarithm problem in a large space by several similar problems in a more tractable search space while retaining full security. On the one hand, this yields for us a general form for two ElGamal variants which are homomorphic with respect to addition: one based on the ElGamal set-up in the multiplicative group $\mathbb{Z}_p^*$ and another using the group of $\mathbb{F}_q$-rational points on an elliptic curve. These schemes are homomorphic with respect to addition in $\mathbb{Z}$. This simple CRT expansion technique has a second application. We show that this technique solves an open problem in the paper of Boneh, et al. [4], alleviating message size limitations on the BGN encryption scheme which was shown to allow homomorphic evaluation of 2-DNF circuits. We finish with a discussion of performance of CRT-based ElGamal, which compares favorably to the leading additively homomorphic scheme of Paillier[3].

2 Background

The aim of this section is to survey existing partially homomorphic encryption schemes with special emphasis on systems that play a role in later sections.

2.1 Overview of Homomorphic Encryption Schemes

Homomorphic properties of several standard public key encryption schemes were recognized early on [2]. Both the RSA and ElGamal encryption schemes were

Scheme	Homomorphism	Computation
Textbook RSA	Multiplicative	Mod. Exp. in Z_{pq}
Textbook ElGamal	Multiplicative	Mod. Exp. in $GF(p)$
Goldwasser Micali [10]	XOR	Mod. Exp. in Z_{pq}
Benaloh [11]	Additive	Mod. Exp. in Z_{pq}
Paillier Scheme [3]	Additive	Mod. Exp. in $Z_{(pq)^2}$
Paillier ECC variations [21]	Additive	Scalar-point mult. in elliptic curves
Naccache-Stern [12]	Additive	Mod. Exp. in Z_{pq}
Kawachi-Tanaka-Xagawa [16]	Additive	Lattice Algebra
Okamoto-Uchiyama [14]	Additive	Mod. Exp. in Z_{p^2q}
Boneh-Goh-Nissim [4]	2-DNF formulas	Mod. Exp. in $Z_{(pq)^2}$, Bilinear Map
Melchor-Gaborit-Herranz [18]	d-op. mult.	Lattice Algebra

Table 1. Survey of partially homomorphic encryption schemes

immediately seen to have homomorphic properties, but only with respect to one operation. Ironically, this aspect of these schemes was largely seen as a weakness rather than an asset. Applications where data is static typically require non-malleable encryption. However, the community has grown to trust the security of these schemes and, recently, the work of Gentry and others demonstrates that, when carefully employed, such homomorphic properties can be quite valuable. Indeed, a number of recent specific applications such as data aggregation in distributed networks [5, 6], electronic voting [7], biometrics [8] and privacy preserving data mining [9] have led to reignited interest in homomorphic schemes.

A list of prominent partially homomorphic schemes is presented in Table 1. One of the earliest discoveries relevant here was the Goldwasser-Micali cryptosystem [10] whose security is based on the quadratic residuosity problem and which allows homomorphic evaluation of a bitwise exclusive-or. This scheme has already been applied to the problem of securing biometric information [8]. Other additive homomorphic encryption schemes that provide semantic security are Benaloh [11], Naccache-Stern [12], Paillier [3], Damgård-Jurik [13], Okamoto-Uchiyama [14] and Boneh-Goh-Nissim [4]. Some additively homomorphic encryption schemes use lattices or linear codes [15–19]. For instance, the lattice-based encryption scheme introduced by Melchor, Gaborit and Herranz [18] allows homomorphic computation of functions expressible as d-operand products of terms, each of which is a sum of inputs.

Of particular interest to us is the Boneh-Goh-Nissim partially homomorphic encryption scheme [4], which allows evaluations of arbitrary 2-DNFs, i.e., functions whose evaluation requires one multiplication per term followed by an arbitrary number of additions of terms. The scheme is based on Paillier's earlier additive partially homomorphic scheme and bilinear pairing. As a consequence, the Boneh-Goh-Nissim scheme allows the secure evaluation of degree-two multivariate polynomials, with dot product computation being a particularly useful primitive arising as a special case. Paillier's scheme is the most efficient among currently known additively homomorphic schemes. In order to compare a specific implementation of our scheme against this system (Sec. 4.2), we review in Section 2.3 some details of the Paillier scheme and an extension by Galbraith [24] to the elliptic curve context. Likewise, we present a brief overview of the Boneh-Goh-Nissim scheme mentioned above in Section 2.4 as we will later use our general setup in Section 3.3 to devise a modification which eliminates its current restriction on output size.

2.2 Homomorphic Properties of ElGamal Encryption

Let G be a finite group and let $g \in G$ be a generator of a cyclic subgroup H of size N. In the standard generic form of ElGamal encryption [20], a user Bob chooses a secret random exponent $k \in \mathbb{Z}_N$ and publishes (g, h) where $h = g^k$ in G; the value $k = \log_g h$ remains secret. A second user, Alice say, who wishes to send a message $m \in H$ to Bob first generates a random exponent ℓ and sends the ordered pair $\mathrm{Enc}_h(m) = (g^\ell, h^\ell m)$ to Bob who, upon receipt of the ordered pair (x, y), computes $m = x^{-k}y$. As is well-known this encryption scheme is homomorphic with respect to multiplication in H: if (x_1, y_1) is a valid encryption for message m_1 and (x_2, y_2) is a valid encryption of another message m_2 (with the same key (g, h)), then (x_1x_2, y_1y_2) is a valid encryption of m_1m_2. We now present two additively homomorphic specializations of this set-up and highlight a fundamental deficiency of such systems.

- **Additive ElGamal encryption using a finite field.** As pointed out by Cramer, et al. [23, Sec. 2.5], among others, one easily obtains an additively homomorphic encryption function by placing plaintexts in the exponent. If $H = G = \langle g \rangle$ is the group of units in some finite field $\mathbb{F}_q$ and $N = q - 1$, then the general setup above is modified to map $a, b \in \mathbb{Z}_N$ to $\mathrm{Enc}_h(g^a)$ and $\mathrm{Enc}_h(g^b)$, respectively, so that componentwise multiplication of ciphertexts yields a valid encryption of g^{a+b}, which is viewed as an encryption of $a + b \in \mathbb{Z}_N$.
- **Elliptic curve version of ElGamal encryption.** Assume E represents an *elliptic curve* over a finite field $\mathbb{F}_q$. We work in the cyclic subgroup $H = \langle P \rangle$ generated by a point P on the curve having order N. Bob chooses a secret random $k \in \mathbb{Z}_N$ and defines $Q = kP$. To encrypt a message m, Alice first encodes it as the point $M = mP$; she then picks a random $\ell \in \mathbb{Z}_N$ and sends $\mathrm{Enc}_k(M) = (\ell P, \ell Q + M)$. Upon receipt of an ordered pair (U, V) of elliptic curve points, the corresponding decryption function computes $\mathrm{Dec}_k(U, V) = M$. This elliptic curve ElGamal scheme is also additively homomorphic: if m_1 and m_2 are mapped to $M_i = m_iP$ and encrypted as $(\ell_iP, \ell_iQ + M_i)$ for $i = 1, 2$, we easily find $\mathrm{Enc}_k(M_1) + \mathrm{Enc}_k(M_2)$ is a valid form for $\mathrm{Enc}_k(M_1 + M_2)$ (with a different randomizer, $\ell_1 + \ell_2$).

Unfortunately, recovering the integer m from the elliptic curve point $M = mP$ in the above scheme requires us to solve an elliptic curve discrete logarithm problem (ECDLP), the hardness of which is the very essence of the security we seek. Since an additively homomorphic scheme of this sort can be quite useful in data aggregation, it has been suggested [6] that some applications warrant this expensive decryption process. We do not find this practical. Likewise, the finite field variant above is hampered by a difficult discrete logarithm computation for proper decryption. As noted in [23], the scheme appears to be useful only if all plaintexts m are limited to a small range. Our first contribution in Section 3 below is to propose a simple tool to overcome these deficiencies for additively homomorphic ElGamal schemes.

2.3 The Paillier Scheme

Paillier [3] proposed a very interesting additive homomorphic encryption scheme based on a decision problem[1] closely related to the problem of deciding n^{th} residues in the ring $\mathbb{Z}_{n^2}$. Let p and q be large primes and set $n = pq$; the group of units in the ring $\mathbb{Z}_n$ has exponent $\lambda = \lambda(n) = \text{lcm}(p-1, q-1)$. The n^{th} roots of unity in $\mathbb{Z}_{n^2}$ are $u = 1 + kn$ $(0 \leq k < n)$ and k is recovered as $L(u) = \frac{u-1}{n}$.

The Paillier's scheme requires a base g satisfying $\gcd(L(g^\lambda \bmod n^2), n) = 1$. Primes p and q and the integer λ are kept private while the public key is (n, g). To encrypt a message $m \in \mathbb{Z}_n$, Alice generates a random r and computes $\text{Enc}(m) = g^m r^n \pmod{n^2}$. Paillier [3] gives evidence that recovering m from $c = \text{Enc}(m)$ without the knowledge of p, q or λ is hard. However, with the knowledge of private key λ (or, equivalently, the factorization $n = pq$), m may be easily recovered from c by computing

$$m = \text{Dec}(c) = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \pmod{n} .$$

The encryption function is additively homomorphic, i.e., $\text{Enc}(m_1)\text{Enc}(m_2) = g^{m_1} r_1^n g^{m_2} r_2^n \pmod{n^2} = \text{Enc}(m_1 + m_2)$. Paillier's scheme requires only simple operations in the encryption, decryption and addition procedures and thus achieves high performance. In addition, Paillier also proposed several modifications to further improve the decryption speed.

In 2000, Paillier [21] proposed several extensions of the above scheme to groups based on elliptic curves. In order to accomodate efficient decryption, the technique requires one to employ elliptic curves over rings of the form $\mathbb{Z}_{n^2}$ or $\mathbb{Z}_{pq}$. These schemes, while similar to the above approach using modular arithmetic, generated certain skepticism due to their reliance on anomalous curves. In 2002, Galbraith [24] published a critique of these elliptic curve-based Paillier schemes, using p-adic methods to show that the second and third schemes are insecure. Galbraith brings Lenstra's elliptic curve factoring method to bear in his attack of the Paillier schemes. In one case, he constructs a quadratic twist of the curve and uses this to factor the public parameter $n = p^2 q$ to recover the secret key $\{p, q\}$. In the other case, a probabilistic argument comes into play, potentially leading to a factorization of the public modulus $n = pq$.

[1] In [3], Paillier defines the "Composite Residuosity Class Problem" and gives compelling evidence for its hardness.

In the same paper, Galbraith proposes an alternative generalization of the original Paillier scheme to elliptic curves, this time using random elliptic curves. While a somewhat convincing argument is given in [24] for the security of this approach, Galbraith observes that the scheme is "mainly of theoretical interest" since the exponent of the elliptic curve group must remain secret, requiring elliptic curve computations over large moduli which are much slower than comparable integer computations.

2.4 The BGN homomorphic scheme

In this section, we review the encryption scheme of Boneh, Goh and Nissim [4], which we henceforth refer to as the BGN scheme.

Groups admitting Bilinear Pairing The BGN scheme uses what is known as a bilinear pairing to effect the required multiplication in evaluating a 2-DNF formula. We use a notation similar to the one in [4] in order to facilitate easier comparison:

1. Let $\mathbb{G}$ and $\mathbb{G}_1$ be two (multiplicative) cyclic groups of finite order n;
2. let g be a generator of $\mathbb{G}$;
3. let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ satisfy three conditions:
   (a) $e(\cdot, \cdot)$ is efficiently computable,
   (b) $e(g, g)$ is a generator of $\mathbb{G}_1$,
   (c) for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}^+$, $e(u^a, v^b) = e(u, v)^{ab}$.

The map e above is a special case of a *bilinear pairing*; one source of such pairings is Tate pairings and another is Weil pairings [30], but this beautiful and complex mathematics is beyond the scope of this paper; we need nothing more here than to know that such maps exist for various groups.

To abstract the construction process in the scheme, define an algorithm $\mathcal{G}$ that given a security parameter $\tau \in \mathbb{Z}^+$ outputs a tuple $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$ where $\mathbb{G}$,$\mathbb{G}_1$ are groups of order $n = q_1 q_2$ and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ is a bilinear pairing. On input τ, algorithm $\mathcal{G}$ works as follows:

1. Generate two random τ-bit primes q_1, q_2 and set $n = q_1 q_2 \in \mathbb{Z}$;
2. Generate two groups $\mathbb{G}$ and $\mathbb{G}_1$ of order n such that there exists a generator g for $\mathbb{G}$ and a bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$;
3. Output $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$.

The BGN Scheme Using the definition of the bilinear pairings and the construction function $\mathcal{G}(\tau)$ presented above, the BGN scheme can be described as follows. (The presentation here differs slightly from that in [4] in order to match our notation and use.)

KeyGen(τ): Given a security parameter $\tau \in \mathbb{Z}^+$, run $\mathcal{G}(\tau)$ to obtain a tuple $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$. Let $n = q_1 q_2$. Pick two random generators $g, u \overset{R}{\leftarrow} \mathbb{G}$ and set $h = u^{q_2}$. Then h is a random generator of the subgroup of $\mathbb{G}$ of order q_1. The public key is $\mathcal{PK} = (n, \mathbb{G}, \mathbb{G}_1, e, g, h)$. The private key is $\mathcal{SK} = (q_1)$.

Encrypt($\mathcal{PK}, m$): We assume the message space consists of integers in the set $\{0, 1, \ldots, T\}$ with $T < q_2$. To encrypt a message m using public key $\mathcal{PK}$, pick a random $r \overset{R}{\leftarrow} \{0, 1, \ldots, n-1\}$ and compute $C = g^m h^r \in \mathbb{G}$. Output C as the ciphertext.

Add($C^{(1)}, C^{(2)}$): Addition is quite straightforward in the scheme. To evaluate the sum of two messages homomorphically, pick a random $r \overset{R}{\leftarrow} \{0, 1, \ldots, n-1\}$ and compute

$$C = C^{(1)} C^{(2)} h^r = g^{m^{(1)}} h^{r^{(1)}} g^{m^{(2)}} h^{r^{(2)}} h^r = g^{m^{(1)}+m^{(2)}} h^{\tilde{r}}$$

Output C as the resulting ciphertext.

Mul($C^{(1)}, C^{(2)}$): The bilinear map is used to perform the one multiplication computation. Set $g_1 = e(g, g)$, and $h_1 = e(g, h)$. Then g_1 is of order n and h_1 is of order q_1. Also, write $h = g^{\alpha q_2}$ for some unknown $\alpha \in \mathbb{Z}$. Then to evaluate the product of two messages homomorphically, pick a random $r \overset{R}{\leftarrow} \{0, 1, \ldots, n-1\}$ and compute

$$\begin{aligned} C &= e(C^{(1)}, C^{(2)}) h_1^r = e(g^{m^{(1)}} h^{r^{(1)}}, g^{m^{(2)}} h^{r^{(2)}}) h_1^r \\ &= g_1^{m^{(1)} m^{(2)}} h_1^{m^{(1)} r^{(2)} + m^{(2)} r^{(1)} + \alpha q_2 r^{(1)} r^{(2)} + r} = g_1^{m^{(1)} m^{(2)}} h_1^{\tilde{r}} \in \mathbb{G}_1 \end{aligned}$$

Output C as the resulting ciphertext. Note that any number of outputs of multiplication gates (i.e., values produced by the bilinear pairing) will have the form $g_1^{m'} h_1^{r'}$ and these can all be viewed as secure encryptions of the corresponding plaintexts m', but instead in the group $\mathbb{G}_1$, where the same additive homomorphic properties hold.

Decrypt($\mathcal{SK}, C$): To decipher C using private key $\mathcal{SK} = (q_1)$, observe that $C^{q_1} = (g^m h^r)^{q_1} = (g^{q_1})^m$. Let $\hat{g} = g^{q_1}$. To recover m, it suffices to compute the discrete log of C^{q_1} base $\hat{g}$. Since $0 \leq m \leq T$ this takes expected time $\tilde{O}(\sqrt{T})$ using Pollard's lambda method [31]. It is clear that the message space of the BGN scheme is limited due to the complexity of this decryption.

3 Our Contribution

We propose a new approach to overcome this discrete logarithm impasse in the additive version of ElGamal encryption and in the decryption step of the BGN schemes. This new CRT-based technique holds promise for making the additively homomorphic schemes described above more practical. The technique also has potential as a building block for a variety of other protocols where one party needs an advantage over another in computing discrete logarithms.

3.1 The CRT-Based ElGamal Scheme

We now present, in generic form, our CRT-based ElGamal scheme $\mathcal{E}$ which we shall henceforth refer to as the CEG Scheme. The CEG scheme, with security parameter λ, is specified using four procedures: KeyGen, Encrypt, Decrypt and Eval as follows.

KeyGen: Choose a group G with subgroup H generated by $g \in G$ and require $|H|$ to have at least one large prime factor. Pick[2] $k \overset{\$}{\leftarrow} [0, 2^\lambda - 1]$. Compute $h = g^k$. Choose $d_i \in \mathbb{Z}^+$ for $i = 1, \ldots, t$ such that $d = \prod d_i < |H|$ and $\gcd(d_i, d_j) = 1$ for $i \neq j$. Set the message space as $\mathcal{M} = \{0, 1, \ldots, N\}$ where $N < d$. The secret key is $\mathcal{SK} = (k)$ and the public key is $\mathcal{PK} = (g, h, \langle d_1, \ldots, d_t \rangle)$.

Encrypt: A message $m \in \mathcal{M}$ is encrypted as the t-tuple of pairs

$$\mathsf{Encrypt}(m) = \langle (g^{\ell_i}, h^{\ell_i} g^{m_i}) \ , \ i = 1, \ldots, t \rangle \ ,$$

where $m_i = m \pmod{d_i}$ and $\ell_i \overset{\$}{\leftarrow} [0, 2^\lambda - 1]$.

Decrypt: A ciphertext $c = \langle (u_i, v_i) \ , \ i = 1, \ldots, t \rangle$ is decrypted as follows.

$$\mathsf{Decrypt}(c) = \mathrm{CRT}^{-1} \left(\langle \log_{\mathrm{g}}(v_i u_i^{-k}) \ , \ i = 1, \ldots, t \rangle \right) \ ,$$

where $\mathrm{CRT}^{-1}(\langle m_i \ , \ i = 1, \ldots, t \rangle) = \sum_{i=1}^{t} m_i \frac{d}{d_i} \left(\frac{d}{d_i}^{-1} \bmod d_i \right) \bmod d$. The function $\log_g(\cdot)$ denotes the discrete logarithm with respect to generator g.

Eval: Given encryptions of S values $m^{(1)}, \ldots, m^{(S)}$, it is straightforward to obtain an encryption of their sum. We simply perform componentwise multiplication in group G. The resulting $2t$-tuple lists t pairs $(g^{L_i}, h^{L_i} g^{\sum m_i^{(r)}})$ where the sum of residues $m_i^{(r)} \equiv m^{(r)} \pmod{d_i}$ suffice to reconstruct the sum of the integers $m^{(r)}$ via CRT^{-1}.

Correctness. The correctness of the scheme follows from the correctness of ElGamal encryption and the correctness of the CRT. The CRT will yield correct results as long as $d > N$. Let $c = \langle (u_i, v_i) \ , \ i = 1, \ldots, t \rangle$ be the entrywise sum of valid ciphertexts c_j whose corresponding plaintexts satisfy $\sum \mathsf{Decrypt}(c_j) < \prod d_i$. Then $\langle m_1, \ldots, m_t \rangle = \langle \log_{\mathrm{g}}(v_i u_i^{-k}) \ , \ i = 1, \ldots, t \rangle$ satisfies $g^{m_i} u_i^k = v_i$. So $m = \mathrm{CRT}^{-1}(\langle m_i \ , \ i = 1, \ldots, t \rangle)$ satisfies $m \bmod d_i = m_i$ as long as $0 \leq m < \prod d_i$.

Efficiency. The CEG scheme converts one carefully constructed DLP in the subgroup $H := \langle g \rangle$ into a sequence of tractable discrete logarithm problems in the same group. The factors d_i of the composite modulus $d = d_1 d_2 \ldots d_t$ are pairwise relatively prime and $[1, \max(d_i)]$ represents a tractable search space (for the DLP). If each d_i is not too large (say, $w = 16$ bits each) Alice may retrieve each integer m_i, even if she must resort to exhaustive search. But she can exploit standard techniques to do better than this; we present some ideas on this step below. However she obtains these logarithms, Alice can then recover the message m via a simple CRT inversion step.

– CEG-$\mathbb{Z}_p$: **Additive CRT-ElGamal Encryption using a Prime Field.** The integer specialization of the generic CEG will be denoted by CEG-$\mathbb{Z}_p$.

[2] The notation $k \overset{\$}{\leftarrow} \mathcal{U}$ stands for k drawn uniformly at random from a universe $\mathcal{U}$.

KeyGen: Choose a large prime p with generator g for $\mathbb{Z}_p^*$; we require $p-1$ to have at least one large prime factor. Pick a random k as above and set $h = g^k$; the integer $\mathcal{SK} = (k)$ remains secret, while $\mathcal{PK} = (g, h, \langle d_1, \ldots, d_t \rangle)$ are made public, where, as in the general case, the message space is $\mathcal{M} = [0, N]$ and we have small coprime integers d_i with $d = \prod d_i > N$.

Encrypt: A message $m \in \mathcal{M}$ is encrypted as

$$\mathsf{Encrypt}(m) = \langle (g^{\ell_i}, h^{\ell_i} g^{m_i}) \ , \ i = 1, \ldots, t \rangle \ ,$$

where $m_i = m \bmod d_i$.

Decrypt: A given ciphertext $c = \langle (u_i, v_i) \ , \ i = 1, \ldots, t \rangle$ is decrypted as

$$\mathsf{Decrypt}(c) = \mathrm{CRT}^{-1} \left(\langle \log_{\mathrm{g}}(v_i u_i^{-k}) \ , \ i = 1, \ldots, t \rangle \right) \ .$$

The function $\log_g(\cdot)$ denotes the discrete logarithm with respect to generator g in $\mathbb{Z}_p^*$.

– CEG-ECC: **Elliptic Curve Version of CRT-ElGamal Encryption:** The elliptic curve specialization of the above scheme will be denoted by CEG-ECC. For illustrative purposes, we concisely present the three procedures as they specialize to elliptic curve encryption.

KeyGen: Choose an elliptic curve E with element $P \in \mathsf{E}$ such that $|\langle P \rangle|$ is a large prime. Pick $k \overset{\$}{\leftarrow} [0, 2^\lambda - 1]$. Compute $Q = kP$. Choose $d_i \in \mathbb{Z}$ for $i = 1, \ldots, t$ as in the general case. The secret key is again $\mathcal{SK} = (k)$ and the public key is $\mathcal{PK} = (P, Q, \langle d_1, \ldots, d_t \rangle)$. The message space is again $\mathcal{M} = [0, N]$.

Encrypt: A message $m \in \mathcal{M}$ is encrypted as $\mathsf{Encrypt}(m) = \langle (\ell_i P, \ell_i Q + m_i P) \ , \ i = 1, \ldots, t \rangle$, where $m_i = m \bmod d_i$ and $\ell_i \overset{\$}{\leftarrow} [0, 2^\lambda - 1]$.

Decrypt: Letting $\log_P(\cdot)$ denote the discrete logarithm in $\langle P \rangle$ with respect to P, ciphertext $c = \langle (A_i, B_i) \ , \ i = 1, \ldots, t \rangle$ is decrypted as $\mathsf{Decrypt}(c) = \mathrm{CRT}^{-1} (\langle \log_{\mathrm{P}}(B_i - kA_i) \ , \ i = 1, \ldots, t \rangle)$.

Semantic Security of CRT-ElGamal The semantic security of our scheme follows from the semantic security of the ElGamal cryptosystem. If the Decisional Diffie Hellman (DDH) assumption holds in a group, then the ElGamal cryptosystem defined over the same group has semantic security[3]. Note that the CEG scheme works by bundling together t independent ElGamal encryptions of small messages. Due to the semantic security of the generic ElGamal cryptosystem [29] neither message sizes nor dependencies between the messages matter.

Proposition 1. *If the DDH assumption holds in group H, then the* CEG *defined on H is semantically secure.*

[3] If, for instance, the group permits a bilinear map then the DDH assumption will not hold.

Proof Sketch: Assume there exists a CEG distinguisher $\mathcal{D}$ with advantage ϵ. Then we can turn this distinguisher into an ElGamal distinguisher $\mathcal{D}'$ for small messages. This is achieved by simply forging two CEG ciphertexts with first components set to the ElGamal ciphertexts and the remaining components filled with the ElGamal encryption of zero.

3.2 Additive Homomorphism

It is not hard to see that CEG is additively homomorphic. In practice, this additive property is limited in two subtle ways:

Overflow in CRT: Assume we are given S inputs $m^{(1)}, m^{(2)}, \ldots, m^{(S)}$ and we wish to obtain an encryption of their sum by utilizing the additive homomorphic property. We simply perform componentwise multiplication in the group G. The resulting $2t$-tuple lists t pairs of the form $(g^{L_i}, h^{L_i} g^{\sum m_i^{(r)}})$. The CRT inversion step computes the isomorphism $\mathbb{Z}_{d_1} \times \cdot \times \mathbb{Z}_{d_t} \to \mathbb{Z}_d$; so, in order to yield a correct decryption result, we must have $\sum m^{(r)} < d$. To make this more concrete, we define CEG parameters as follows. Let w be a given word size (e.g., $w = 16, 32$); let $d_1, \ldots, d_t$ be t distinct w-bit primes. Then $d = d_1 \cdots d_t < (2^w)^t$. On the other hand, assume each operand $m^{(j)}$ fits into a W-bit word and that the maximum number of additions to be supported is S. Then $\sum m^{(r)} \leq S\ 2^W$. So, with these parameters, we may precisely state necessary and sufficient conditions on our parameters for overflow prevention: $2^{wt} > d \geq S\ 2^W$. Therefore, given w, W and S, we can determine the required number of components in the CRT expansion by picking the smallest integer t satisfying $t > (W + \log_2(S))/w$.

Tractability of $\mathsf{Decrypt}$: The computational bottleneck for $\mathsf{Decrypt}$ is still the computation of t discrete logarithms. This will give us the main restriction on the CEG scheme. In order to make each DLP tractable, we must restrict how large each prime d_i can be. But S values $m_i^{(j)} \in \{0, \ldots, d_i - 1\}$ are being summed without wraparound and this can result in a value as large as Sd_i. So, if we consider a search space of size 2^u manageable, we need to ensure that $S \cdot \max(d_i) < 2^u$, which can be interpreted as a limit on the number of operands that can be handled in a single sum.
For example, if $u = 60$ and $w = 16$ (i.e., $d_i < 2^{16}$ for all i), then we can sum $S = 2^{44}$ operands and still have an acceptable search space. Alternatively, we can increase S by choosing smaller d_i; but this comes at the expense of a larger t value. (Security requirements require d large.) For instance, choosing very small primes $d_i < 2^8$ enables us to increase the number of operands to roughly $S \approx 2^{52}$.

As we have seen, we have various tradeoffs to consider. For small S and moderate d_i, the DLP search is tractable. As the primes d_i are made smaller, we accomodate more summands but the necessary increase in t leads to longer encryptions and increased computation per encryption and per ciphertext addition. Clearly there is room for further research here; for instance, intelligent pre-computation can further reduce decryption time at the expense of several gigabytes of storage to accomodate a lookup table.

3.3 Applying CRT to the BGN homomorphic scheme

As discussed earlier in Section 2.4, the (output) message size of the BGN scheme is limited since decryption requires a discrete logarithm computation. In fact, the BGN paper [4] leaves the message size restriction as an open problem. We find that the application of CRT presents a solution to this problem, i.e. we employ CRT to break large messages into smaller pieces and then encrypt the smaller pieces using the BGN scheme. Since the BGN scheme has semantic security, the overall scheme will still be semantically secure even with smaller message sizes. We present the CRT-BGN scheme first and discuss its efficiency later.

KeyGen(τ): Given a security parameter $\tau \in \mathbb{Z}^+$, run $\mathcal{G}(\tau)$ to obtain a tuple $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$. Let $n = q_1 q_2$. Pick two random generators $g, u \xleftarrow{R} \mathbb{G}$ and set $h = u^{q_2}$. Then h is a random generator of the subgroup of $\mathbb{G}$ of order q_1. Suppose the message space consists of integers in the set $\{0, 1, \dots, N\}$. Choose $d_i \in \mathbb{Z}$ for $i = 1, \dots, t$ such that $d_i < T$, $d = \prod d_i > N$ and $\gcd(d_i, d_j) = 1$ for $i \neq j$. The public key $\mathcal{PK} = (n, \mathbb{G} - \gtrdot Ↄ \ltimes, \mathbb{G}_1, e, g, h, \langle d_1, \dots, d_t \rangle)$. The private key is $\mathcal{SK} = (q_1)$.

Encrypt($\mathcal{PK}, M$): We will encrypt the message with t tuples of BGN scheme. The message space for each tuple will be $\{0, 1, \dots, T\}, T < q_2$. To encrypt a message m using public key $\mathcal{PK}$, pick random variables $r_i \xleftarrow{R} \{0, 1, \dots, n-1\}$ and compute t-tuple of pairs

$$C = \langle C_i \ , \ i = 1, \dots, t \rangle = \langle g^{m_i} h^{r_i} \in \mathbb{G} \ , \ i = 1, \dots, t \rangle \ .$$

where $m_i = m \pmod{d_i}$. Output C as the ciphertext.

Add($C^{(1)}, C^{(2)}$): The addition is done pairwise for the t-tuples. To evaluate the sum of two messages homomorphically, pick random variables $r_i \xleftarrow{R} \{0, 1, \dots, n-1\}$ and compute

$$\begin{aligned} C &= \langle C_i^{(1)} C_i^{(2)} h^{r_i} \ , \ i = 1, \dots, t \rangle = \langle g^{m_i^{(1)}} h^{r_i^{(1)}} g^{m_i^{(2)}} h^{r_i^{(2)}} h_i^{r} \ , \ i = 1, \dots, t \rangle \\ &= \langle g^{m_i^{(1)} + m_i^{(2)}} h^{r_i^{(1)} + r_i^{(2)} + r_i} \ , \ i = 1, \dots, t \rangle = \langle g^{m_i^{(1)} + m_i^{(2)}} h^{\tilde{r_i}} \ , \ i = 1, \dots, t \rangle \ . \end{aligned}$$

Output C as the resulting ciphertext. After the multiplication, the addition can be evaluated in the same way using g_1 and h_1.

Mul($C^{(1)}, C^{(2)}$): Similarly, the multiplication is done pairwise for the t-tuples. Set $g_1 = e(g, g)$, and $h_1 = e(g, h)$. Then g_1 is of order n and h_1 is of order q_1. Also, write $h = g^{\alpha q_2}$ for some unknown $\alpha \in \mathbb{Z}$. Then to evaluate the product of two message homomorphically, pick random variables $r_i \xleftarrow{R} \{0, 1, \dots, n-1\}$

and compute

$$\begin{aligned} C &= \langle e(C_i^{(1)}, C_i^{(2)}) h_1^{r_i} \ , \ i = 1, \ldots, t \rangle \\ &= \langle e(g^{m_i^{(1)}} h^{r_i^{(1)}}, g^{m_i^{(2)}} h^{r_i^{(2)}}) h_1^{r_i} \ , \ i = 1, \ldots, t \rangle \\ &= \langle g_1^{m_i^{(1)} m_i^{(2)}} h_1^{m_i^{(1)} r_i^{(2)} + m_i^{(2)} r_i^{(1)} + \alpha q_2 r_i^{(1)} r_i^{(2)} + r_i} \ , \ i = 1, \ldots, t \rangle \\ &= \langle g_1^{m_i^{(1)} m_i^{(2)}} h_1^{\tilde{r_i}} \in \mathbb{G}_1 \ , \ i = 1, \ldots, t \rangle \ . \end{aligned}$$

Output C as the resulting ciphertext.

Decrypt($\mathcal{SK}, C$): To decrypt a ciphertext C using a private key $\mathcal{SK} = (q_1)$, for each tuple we have: $C_i^{q_1} = (g^{m_i} h^{r_i})^{q_1} = (g^{q_1})^{m_i}$ where the g, h here could be g_1, h_i if the multiplication is computed.
Let $\hat{g} = g^{q_1}$. To recover m_i, it suffices to compute t tuples of the discrete log of $C_i^{q_1}$ base $\hat{g}$. Since $0 \leq m_i \leq T$ this takes expected time $\tilde{O}(\sqrt{T})$ using Pollard's lambda method. After recovering all the m_i values, the plaintext can be reconstructed as $m = \mathrm{CRT}^{-1}(m_i \mod d_i \ , \ i = 1, \ldots, t)$.

Correctness: The correctness of the CRT-BGN scheme follows from the correctness of the BGN and CRT schemes. The correctness of each component in CRT comes from the property of the multiplicative cyclic group. As long as the sub-result of each component is smaller than T, the residues can be efficiently decrypted. After the recovery of each m_i, if the result of the computation satisfies $m < d = \prod d_i$, then m can be correctly recovered by the definition of CRT.
Efficiency: The main efficiency concern in the CRT-BGN scheme is the latency of decryption since it requires DLP computations. In comparison, the cost of the inverse CRT computation is negligible. The scheme is constructed using a composite modulus $d > N$ with factors $d_1, d_2 \ldots d_t$, where d_i are pairwise relatively prime. The selection of the size of d_i and the number of components t is limited by the size of the message space N. We refer to the largest d_i as $\tilde{d} = \max\{d_i, i = 1 \ldots t\}$. Then it is clear that $\tilde{d} > (N)^{1/t}$. On the other hand, suppose A additions will be evaluated before the multiplication and B additions after, then the message space T for each component has to satisfy $T > A^2 B \tilde{d}^2$ for correct decryption.

By combining the two equations we obtain $T > A^2 B (N)^{2/t}$. If T is not too large, one may recover the plaintext efficiently even if exhaustive search is used. Pollard's lambda method [31] may cut the search time for each CRT component from $O(T)$ to $\tilde{O}(\sqrt{T})$. Therefore, the complexity for decrypting all the t components will be $\tilde{O}(tAN^{1/t}\sqrt{B})$ compared to $\tilde{O}(\sqrt{N})$ in the original BGN scheme. By this we increase the tractable message space significantly. We note that if we want to increase the message space N while keeping the decryption complexity for each component $\tilde{O}(AN^{1/t}\sqrt{B})$ the same, the number of components t need grow only logarithmically with the size of the message space.

Security Analysis of CRT-BGN The security of the BGN scheme relies on the subgroup decision assumption defined as follows.

Definition 1. Subgroup Decision Problem: *Given $(n, \mathbb{G}, \mathbb{G}_1, e)$ where e is a bilinear pairing from $\mathbb{G} \times \mathbb{G}$ to $\mathbb{G}_1$ (two cyclic groups of order $n = q_1q_2$) and an element $x \in \mathbb{G}$, output '1' if the order of x is q_1. Otherwise output '0'.*

When q_1 and q_2 are τ-bit primes, the advantage of any algorithm $\mathcal{A}$ in solving the subgroup decision problem is:

$$SD\text{-}Adv_{\mathcal{A}}(\tau) = |Pr[\mathcal{A}(n, \mathbb{G}, \mathbb{G}_1, e, x) = 1] - Pr[\mathcal{A}(n, \mathbb{G}, \mathbb{G}_1, e, x^{q_2}) = 1]|$$

where the probability is over x chosen uniformly from $\mathbb{G}$.

Using this and referring to the subgroup decision problem, the definition of the subgroup decision assumption can be described as follows:

Definition 2. Subgroup Decision Assumption: *Given algorithm $\mathcal{G}$ generating bilinear group $(n = q_1q_2, \mathbb{G}, \mathbb{G}_1, e)$, we say $\mathcal{G}$ satisfies the subgroup decision assumption if for any polynomial time algorithm $\mathcal{A}$ we have that $SD\text{-}Adv_{\mathcal{A}}(\tau)$ is a negligible function in τ.*

Similar to the BGN scheme, the semantic security of the system with CRT can be proved under the subgroup decision assumption.

Theorem 1. *The CRT-BGN scheme is semantically secure assuming $\mathcal{G}$ satisfies the subgroup decision assumption.*

Proof. Suppose a polynomial time algorithm $\mathcal{B}$ breaks the semantic security of the CRT-BGN scheme with advantage $\epsilon(\tau)$. We can construct an algorithm $\mathcal{A}$ that breaks the subgroup decision assumption with the same advantage.

Given $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$ generated by $\mathcal{G}$ and for an input $x \in \mathbb{G}$, we execute the following procedure:

1. $\mathcal{A}$ receives $(n = q_1q_2, \mathbb{G}, \mathbb{G}_1, e, x)$ as input ($\mathcal{A}$ does not have access to q_1 and q_2). $\mathcal{A}$ then runs the KeyGen process. After calling $\mathcal{G}(\tau)$ to obtain a tuple $(q'_1, q'_2, \mathbb{G}', \mathbb{G}'_1, e')$ and set $n' = q'_1q'_2$, $\mathcal{A}$ substitutes $n', \mathbb{G}', \mathbb{G}'_1, e'$ with the corresponding respective input values. Since the input is also generated using the same function $\mathcal{G}(\tau)$, the bilinear groups should follow the same distribution. $\mathcal{A}$ can then continue the KeyGen process. However, $\mathcal{A}$ directly picks x as the h in the process instead of computing $h = u^{q_2}$. Since the other steps of the KeyGen process are independent of the unknown parameters q_1 and q_2, $\mathcal{A}$ can complete KeyGen and output $(n, \mathbb{G}, \mathbb{G}_1, e, g, x, \langle d_1, \ldots, d_t \rangle)$. Note that if x is of order q_1, it will follow the same distribution as h, and the output becomes a valid public key for the CRT-BGN scheme.
2. $\mathcal{B}$ outputs two valid messages $m^{(0)}, m^{(1)}$ to $\mathcal{A}$. $\mathcal{A}$ picks a random $b \overset{\$}{\leftarrow} \{0, 1\}$ and encrypts the message $m^{(b)}$ using the CRT-BGN scheme and sends back the ciphertext.
3. $\mathcal{B}$ outputs its guess $b' \in \{0, 1\}$ for b. If $b = b'$, $\mathcal{A}$ outputs 1; otherwise $\mathcal{A}$ outputs 0.

If x is uniform in $\mathbb{G}$, then with high probability, x has order q_1q_2 (so the system is not a working CRT-BGN system). The ciphertext will then also be uniformly distributed in $\mathbb{G}$, independent of x. Algorithm $\mathcal{B}$ will then have no non-negligible advantage. Therefore, the probability of a correct guess will be 1/2. If x is of order q_1, the system works as a CRT-BGN system. By our hypothesis on $\mathcal{B}$, we know that $Pr[b = b'] > 1/2 + \epsilon(\tau)$. Therefore, $\mathit{SD\text{-}Adv}_{\mathcal{A}}(\tau) > \epsilon(\tau)$. $\mathcal{A}$ breaks the subgroup decision assumption with advantage $\epsilon(\tau)$.

4 Performance

4.1 Optimizations for the CEG-CRT Scheme

For the sake of simplicity we focus our attention on the elliptic curve specialization CEG-ECC. Similar optimizations may be applied to the other specializations.

Encryption. There are number of optimization techniques we can utilize in the implementation of the CEG scheme. Recall that each component of the CEG ciphertext is in the form $(\ell P, \ell Q + mP)$. The encryption procedure for each component involves only three point-scalar multiplications and one point addition neglecting the generation of the random integer ℓ. This simple approach improves the speed of the encryption procedure. In addition, the encryption can be further sped up using Shamir's Trick. Since the plaintext m is much smaller than the random number ℓ, the latency of the encryption procedure is dominated by the computation of ℓP and ℓQ. For different CRT components, we would compute different $\ell_i P$ and $\ell_i Q$ however with the same P and Q, this is ideally suited for Shamir's Trick [20]. With Shamir's Trick, the complexity of CEG encryption for all t components would become $c \cdot \frac{t+2}{3t}$ instead of ct, where c represents the time required for the encryption of a single component.

Decryption. The CEG decryption may use the Pollard Kangaroo Algorithm to solve the discrete logarithm problem. More specifically, in the CEG scheme, if we want to solve for m given $C_0 = mP$, where $m \in [0, b]$, we generate random walks from $T_0 = bP$ with iterations defined as $T_i = x_i P + T_{i-1}, x_i = f(T_{i-1})$ where f is a hash function. After a number of steps we place a *trap* $T = (b + \sum x_i)P$. Then we start a similar procedure from $C_0 = mP$ and get $C_i = y_j P + C_{j-1}, y_j = f(C_{j-1})$. If the trap is placed after sufficiently many steps, the second run (the kangaroo) will collide with the trap with a high probability. When this collision happens, we can then solve for m from $m = b + \sum x_i - \sum y_j$. In our experiment the steps x_i and y_j are generated with an average size of $0.5\sqrt{b}$ and the trap is placed after $\sqrt{b}$ steps. It is clear that the expectation of the trap position would be $1.5b$. Therefore, $2\sqrt{b}$ steps are expected before the kangaroo is caught by the trap. The complexity of this algorithm is $O(\sqrt{b})$.

We can exploit the fact that we need to perform t parallel DLP computations to reduce the overall decryption complexity. After the accumulation of S operands with word size w the upper limit of the value of m would be $2^{\frac{w+log(S)}{2}}$. Therefore, the complexity of recovering each CRT component using the Pollard Kangaroo algorithm will be $O(2^{\frac{w+log(S)}{2}})$. However, in recovering the various CRT components, we are solving discrete logarithm problems in the same known group. Therefore, only one trap is required. Therefore, the first phase of the Pollard Kangaroo procedure needs only to be executed for the first CRT component; for all remaining components we need only find the collision. This approach saves about 1/3 of the steps. Therefore, the complexity for all t components together can be reduced to $O(\frac{2t+1}{3} \cdot 2^{\frac{w+log(S)}{2}})$.

Decryption with Precomputation. Since we envision a scheme where the same group, the same generator and the same primes d_i will be used repeatedly, we can significantly exploit precomputation techniques to speed up the Pollard Kangaroo Algorithm. For instance, the "trap" can be pre-computed as it is independent of the ciphertext. Also, when we are computing $C_i = y_j P + C_{j-1}, y_j = f(C_{j-1})$ during the second run, the costly multiplication $y_j P$ can be significantly speed up by using table lookup; $\sqrt{b}$ entries are required for such a table.

Moreover, since the search space of our scheme is relatively small, we can even get rid of the Pollard Kangaroo Algorithm and directly apply table lookup techniques on the DLP computation. For instance, a naive approach is to store all $b = S \cdot 2^w$ possible pairs (i, g^i) in a table sorted by the second coordinate. At the expense of substantial storage this reduces the DLP to t lookup operations requiring $tw \log_2(S)$ computational steps. A more reasonable compromise is to precompute a fraction of the search space, say z out of b evenly spaced points. The table lookup remains negligible while the search space is reduced to only b/z. After this many iterations $y \mapsto y \cdot g$ we are assured a collision and the table lookup completes the computation of our DLP. For example, for 160-bit CEG-ECC with $b = 2^{32}$ and $z = 2^{16}$, we will need a table of $z = 2^{16}$ rows with each row contains the 32-bit i and the 160-bit g^i (representing elliptic curve points by their x coordinate). (only x part of the points is more than enough.). Then the lookup table contains about $192 \cdot 2^{16}$ bits$\approx$ 1.6 Mbytes. [4] The number of components will not affect the storage overhead as all t components can share the same table.

4.2 Implementation Results

To evaluate the performance of the CEG scheme, we implemented one CRT component and used the measured performance to estimate the addition, encryption and decryption speed of the full CEG scheme. We also implemented the standard version of Paillier's scheme (Page 7, [3]) for comparison. The implementation is realized on an Intel Core i5 2.4GHz CPU.

In the implementation for both our CEG schemes and Paillier's scheme we made use of the Crypto++ library [26]. The Crypto++ library is modestly optimized. Therefore, the performance of Paillier's scheme may be worse than some optimized implementations [27]. To be thorough we also present results for a more efficient implementation of Paillier's scheme using the MPIR library [28]. But we note that the first implementation may be the most natural one

[4] In practice, we do not need to store the entire operand, but sufficiently many bits, e.g. 64-bits, enough to uniquely identify the point with high confidence.

to compare against as our CEG scheme implementations are using the same optimization level.

Several parameter choices such as w, W, S and t will affect the performance of the CEG scheme. The efficiency of Paillier's scheme is not tied to these parameters. The performance for the original CEG-ECC scheme, CEG-$\mathbb{Z}_p$ scheme and Paillier's scheme for various numbers of summands with various word sizes is given in Table 2. We chose a 224-bit NIST curve for the CEG-ECC scheme and selected a Paillier scheme with roughly equivalent security level of 2048-bits [25]. Our CEG-$\mathbb{Z}_p$ scheme implementation also uses a 2048 bit prime number.

Parameters	Addition	Encryption		Decryption	
		Normal	Shamir's trick	Pollard Kangaroo	Precomputation
CEG-$\mathbb{Z}_p$-2048					
$w = 8, S = 2^{24}, t = 7$	0.22 ms	115.71 ms		79.63 sec	1.35 sec (0.75 Mbytes)
CEG-ECC-224					
$w = 8,\ S = 2^{24}, t = 7$	0.22 ms	12.37 ms	5.30 ms	99.61 sec	2.76 sec (0.75 Mbytes)
$w = 16, S = 2^{24}, t = 4$	0.12 ms	7.07 ms	3.53 ms	15.93 min	25.17 sec (12 Mbytes)
$w = 8,\ S = 2^{40}, t = 9$	0.28 ms	15.90 ms	6.48 ms	8.97 hours	15.10 min (192 Mbytes)
Paillier - Crypto++					
$n = 2048$	0.028 ms	29.60 ms		28.10 ms	
Paillier - MPIR					
$n = 2048$	0.013 ms	25.90 ms		24.90 ms	

Table 2. Performance comparison of CEG with Paillier's Scheme

From Table 2 we can see that the CEG-ECC scheme is about 4 times faster than the Paillier scheme in encryption at comparable security levels. In this simplest approach, the decryption performance of CEG is significantly worse than that of Paillier. However, with precomputation the decryption performance may be improved. The precomputation tables were fixed to have $\sqrt{b} = \sqrt{S2^W}$ rows. For instance, for a very modest precomputation table of 0.75 Mbytes we can reduce the decryption time to less than 3 seconds[5]. In many applications of homomorphic encryption schemes the encryption and evaluation speeds matter more than the decryption speed since typically decryption is performed only once after the computations are completed.

5 Conclusion

We proposed a simple solution to the pervasive bottleneck in additive homomorphic encryption schemes based on the hardness of the DLP. We employed the CRT to replace one discrete logarithm problem in a large space by several similar problems in a more tractable search space while retaining full security. This yields, the first practical elliptic curve-based additive homomorphic encryption

[5] In the table, we assume that 64 bits is sufficient to uniquely identify the points in the precomputation table.

scheme. More generally, our CRT technique makes discrete logarithm problems asymmetric, thereby lending itself as a tool to build a number of practical DLP based additive homomorphic schemes.

As an example, our CEG-ECC scheme is shown to be almost 4 times faster than Paillier scheme on the encryption side while maintaining comparable security. This maybe useful in applications where encryption is used much more often than decryption. A key feature of the proposed scheme is its flexible message space. Solely for the sake of security, the Paillier encryption scheme always requires 4096-bit operations, even for single bit computations. In contrast, the CEG-ECC scheme can be customized to meet the message space demands of any application; the lower limit is set by the difficulty of the ECC discrete logarithm problem. But the message space of the proposed scheme can be easily expanded by simply adding more CRT components. As the number of components grows only logarithmically with the message space, the performance impact is quite manageable.

We also found the CRT approach valuable as a way to solve an open problem in [4] where Boneh, Goh and Nissim describe a scheme which homomorphically evaluates 2-DNF formulas. In their paper, the need to perform a discrete logarithm as part of the decryption process seemed to place a strict limitation on the size of their message space. The Chinese Remainder Theorem variant CRT-BGN maintains security while replacing this hard discrete logarithm computation with t discrete logarithm problems in a completely manageable search space.

References

1. C. Gentry, Fully homomorphic encryption using ideal lattices, Symposium on the Theory of Computing (STOC), 2009, pp. 169-178.
2. R.L. Rivest, L. Adleman, and M.L. Dertouzos. On data banks and privacy homomorphisms. In Foundations of Secure Computation, 1978.
3. P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in Advances in Cryptology EUROCRYPT'99, LNCS 1592, pp. 223–238, Springer, New York, NY, USA, 1999.
4. D. Boneh, E. Goh, K. Nissim, Evaluating 2-DNF Formulas on Ciphertexts, TCC '05, LNCS 3378, pp. 325-341, 2005.
5. E. Mykletun, J. Girao, and D. Westhoff. Public Key Based Cryptoschemes for Data Concealment in Wireless Sensor Networks. In IEEE Int. Conference on Communications ICC, Istanbul, Turkey, June 2006.
6. Osman Ugus, Dirk Westhoff, Ralf Laue, Abdulhadi Shoufan, Sorin A. Huss, Optimized Implementation of Elliptic Curve Based Additive Homomorphic Encryption for Wireless Sensor Networks. WESS '07, Salzburg, Austria, 2007.
7. Aggelos Kiayias, Moti Yung, Tree-Homomorphic Encryption and Scalable Hierarchical Secret-Ballot Elections. Financial Cryptography 2010: pp. 257–271.
8. Julien Bringer, Hervé Chabanne, Malika Izabachéne, David Pointcheval, Qiang Tang and Sébastien Zimmer, An Application of the Goldwasser-Micali Cryptosystem to Biometric Authentication, Information Security and Privacy, LNCS 4586, pp. 96–106, 2007.

9. M. Kantarcioglu, *Privacy-preserving distributed data mining and processing on horizontally partitioned data*, PhD. dissertation, Department of Computer Science, Purdue University, 2005.
10. S. Goldwasser, S. Micali, Probabilistic Encryption, J. Comp. Sys. Sci., 28, pp. 270–299, 1984.
11. Josh Benaloh, Dense Probabilistic Encryption, SAC 94, pages 120–128, 1994.
12. D. Naccache, J. Stern. A New Public Key Cryptosystem Based on Higher Residues. Proceedings of the 5th ACM CCS, pages 59–66, 1998.
13. I. Damgård and M. Jurik. A Length-Flexible Threshold Cryptosystem with Applications. ACISP '03, pp. 350–356.
14. T. Okamoto and S. Uchiyama. A New Public-Key Cryptosystem as Secure as Factoring. Eurocrypt' 08, LNCS 1403, pp. 308-318, 1998.
15. C. Peikert and B. Waters. Lossy Trapdoor Functions and Their Applications. STOC 08, pp. 187–196.
16. A. Kawachi, K. Tanaka, K. Xagawa. Multi-bit cryptosystems based on lattice problems. PKC '07, pp. 315–329.
17. C.A. Melchor, G. Castagnos, and P. Gaborit. Lattice-based homomorphic encryption of vector spaces. ISIT '08, pp. 1858–1862.
18. Carlos Aguilar Melchor and Philippe Gaborit, Javier Herranz, Additively Homomorphic Encryption with d-Operand Multiplications CRYPTO 2010, pp. 138–154, 2010.
19. F. Armknecht and A.-R. Sadeghi. A new approach for algebraically homomorphic encryption. Eprint 2008/422.
20. T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory, IT- 31(4):469–472, 1985.
21. P. Paillier, Trapdooring discrete logarithms on elliptic curves over rings, ASIACRYPT 2000, LNCS 1976, pp. 573–584. 2000.
22. Craig Gentry Shai Halevi, Implementing Gentry's Fully-Homomorphic Encryption Scheme Preliminary Report, August 5, 2010. `https://researcher.ibm.com/researcher/files/us-shaih/fhe-implementation.pdf`
23. Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. EUROCRYPT'97, LNCS 1233, pp. 103–118, 1997.
24. S. Galbraith, Elliptic curve paillier schemes, Journal of Cryptology, vol. 15, no. 2, pp. 129–138, 2002.
25. Arjen K. Lenstra, Eric R. Verheul, Selecting Cryptographic Key Sizes. Vol 14, No 4, Journal of cryptology, Springer, 2001. pp. 255–293.
26. Crypto++ Library 5.6.1, http://www.cryptopp.com/
27. Frederiksen, T.K., A Practical Implementation of Regev's LWE-based Cryptosystem, 2010.
28. MPIR 2.4.0, http://www.mpir.org/
29. Y. Tsiounis and M. Yung, On the security of ElGamal based encryption. pp. 117–134 PKC98 LNCS 1431, 1998.
30. Victor S. Miller, The Weil Pairing, and Its Efficient Calculation, Journal of Cryptology 17(4):235-261, Springer, 2004
31. Menezes, A.J. and Van Oorschot, P.C. and Vanstone, S.A., Handbook of applied cryptography, CRC Press, 2005.

Invited Paper: A New Masking Scheme for Side-Channel Protection of the AES*

Julien Bringer[1], Hervé Chabanne[1,2], and Thanh Ha Le[1]

[1] Morpho, Safran group
[2] Télécom ParisTech

Abstract. We introduce a new protection of data against Side Channel Analysis (SCA) based on wire-tap codes that we illustrate here on the AES cipher. We point out that it brings two novel features: the possibility to unmask without the knowledge of the mask and its capability to detect some faults.

Keywords. Side Channel Analysis, Wire-tap codes, AES cipher.

1 Introduction

Side-Channel Analysis (SCA) has been introduced to recover secret data that are inputs of a cryptographic algorithm. SCA exploits the physical characteristics of the physical token where the algorithm is implemented. Namely, it exploits its leakage (timing, power consumption, electromagnetic emanations) made during the execution of this algorithm. Statistical treatments are then performed to extract secret data from all the captures.

To enhance security against higher-order DPA, a few masking techniques are suggested. In particular, it is possible to use several random additive masks at the same time [11] to ensure provable security against higher-order DPA (d masks for security against $(d-1)$th-order DPA). Affine masking is another solution that was first discussed in [14] and recently further studied in [3] that combines additive masking and multiplicative masking (either as a linear multiplication by a matrix or as a field multiplication). Compared with multiple padding, [3] shows that the affine masking is provably secure only at first order DPA but that it can achieve higher order resistance against DPA in practice (i.e. an important number of consumption traces is needed). Affine masking is based on a mapping $x \in \{0,1\}^k \mapsto x.L \oplus m$ where m is a random element in $\{0,1\}^k$ and L is either a random linear bijection in $\{0,1\}^k$ [14] or a random element in $GF(2^k)$ [3]. Note that many other masking methods, not based directly on additive or multiplicative masking, have been designed for block ciphers implementations, in particular for the AES, see for instance [1, 7, 2, 10, 4]. Nonetheless, improvements or new methods are still requested to increase the resistance against more and more advanced high-order attacks.

* This work has been partially funded by the JST/ANR SPACES (Security evaluation of Physically Attacked Cryptoprocessors in Embedded Systems) project.

In this paper, we consider another secrecy system, namely the wire-tap channel. [15] has shown a way to convey information confidentially whenever a receiver enjoys a better channel than its adversary does.

We introduce a new masked version of the AES following to this principle of wire-tap channel. Our masking method is based on the mapping $x \in \{0,1\}^k \mapsto x.L \oplus c \ \in \{0,1\}^n$ where c is a random codeword from a well-chosen code of length n and L is a linear transformation from $\{0,1\}^k$ into a subspace of $\{0,1\}^n$. This method can be interpreted as an improvement of the affine masking to the wire-tap channel context. Our proposal has the property that the length of the message is expanded during the protection. Due to wire-tap secrecy this gives us a secure Simple Power Analysis resistant masking, and we show that it also achieves first and second-order resistance in practice. As a side effect of the presence of error correcting codewords, we can efficiently detect the presence of some faults during an execution of the algorithm.

Our protection technique has the unique property that one can unmask without the need to use the mask itself. In [13], the authors show how to mount an High Order DPA attack on the AES using at the same time both the masked value and its mask. With respect to this result, we can stress the importance of this asset of our proposal. To the best of our knowledge, our countermeasure is the sole masking solution with this feature.

2 Wire-Tap Codes

The wire-tap channel problem was first introduced by Wyner [15] in 1975 and later extended in [9]. The context is a transmission channel (noisy or not) between two parties Alice and Bob where an eavesdropper Eve can only see a degraded version of the transmitted messages. Assume that the channel between Alice and Bob is noiseless, then it means that the transmission channel from Alice to Eve (or from Bob to Eve) is either with noise or with erasures. We consider here the second case, i.e. the erasure wire-tap channel [9] with a probability of ϵ to have an erasure (this is a Binary Erasure Channel BEC(ϵ)).

The principle to encode a secret message in the context of an erasure wire-tap channel is the following. To transmit k-bit messages, a binary linear code C of length n is chosen and each message is associated to a coset of C in such a way that it is not possible for Eve to gain information on the transmitted message when observing at most $\mu < n$ bits of the encoded message. More precisely, let

- G be a generator matrix of size $(n-k) \times n$ of C, with rows $(g_1, \dots, g_{n-k})$,
- $L = (l_1, ..., l_k)$ be a family of k linearly independent vectors from $\{0,1\}^n$, not in C,
- $m = (m_1, ..., m_{n-k})$ be a uniformly random vector of $(n-k)$ bits,

then a message $x = (x_1, \dots, x_k)$ in $\{0,1\}^k$ is encoded as $z = x.L \oplus mG$ where $x.L = x_1 l_1 \oplus \cdots \oplus x_k l_k \in \{0,1\}^n$ and $c = mG$ is the random codeword related to m ($c = m_1 g_1 \oplus \cdots \oplus m_{n-k} g_{n-k}$). Here L and G are in fact chosen such that $(g_1, \dots, g_{n-k}, l_1, \dots, l_k)$ span the entire space vector $\{0,1\}^n$.

Let H be the parity matrix of C, i.e. the $k \times n$ matrix such that for all $c \in C$, $Hc^T = 0$.

In this construction, the information rate of the channel is $R = k/n$ (1 minus the information rate of C). Tolerating an information rate below 1 enables to resist to the leakage of some bits to an eavesdropper.

Indeed, consider the received message $\tilde{z}$ by Eve. Eve has to search which coset of C corresponds to $\tilde{z}$. A coset will correspond to $\tilde{z}$ if it contains at least one vector that is equal to $\tilde{z} \in \{0,1\}^n$ in the unerased positions. The maximum value for the total number of cosets of C that correspond to $\tilde{z}$ is 2^k in this construction. When this is reached for any input message, this means that the conditional entropy $\mathrm{H}(X|\tilde{Z} = \tilde{z})$ is equal to the entropy of X, i.e. k. This leads to perfect secrecy. A way to ensure this condition is given by the following result.

Lemma 1 ([8, 12]). *Let C be a linear code with a parity matrix $H = \begin{pmatrix} h_1 \cdots h_n \end{pmatrix}$ of size $k \times n$ where h_i is the i-th column of H. Consider an eavesdropper's observation $\tilde{z} \in \{0,1,?\}^n$ with $n - \mu$ erased positions given by $\{i \text{ s.t. } z_i = ?\} = \{i_1, i_2, \ldots, i_{n-\mu}\}$. Then the original message x remains perfectly secret if and only if the sub-matrix $\tilde{H} = \begin{pmatrix} h_{i_1} \cdots h_{i_{n-\mu}} \end{pmatrix}$ is of rank k.*

Therefore, to achieve perfect secrecy in the above construction against eavesdropping of any μ bits of the transmitted message, it is necessary and sufficient that the parity matrix H of the code C satisfies that all of its $k \times (n - \mu)$ submatrices have rank k. It is known [5] that this is equivalent to having a minimum distance equal to $\mu + 1$ for the code generated by H, i.e. for the dual code of C. This implies that $\mu \leq n - k$ and the optimal choice for given parameters n, k is to take for C the dual code of an $[n, k, d]$ binary linear code with the greatest possible minimum distance d.

Various constructions for efficient wire-tap channels are studied under the constraints of high information rate, encoding/decoding performances and approaching secrecy capacity (see for instance [12]). In our case, we will consider small values for n and k so we will rely on more classical code constructions.

3 Protecting AES with Wire-Tap Codes

In the following, we explain how to protect a block cipher algorithm like AES against side-channel analysis thanks to wire-tap codes.

3.1 Brief description of AES

AES [6] takes as input a 16-byte block and consists mainly in the iteration of a round. The 16-byte block is represented as a 4×4 square called a *state* and is subject to the following operations. `AddRoundKey`: this operation adds a round key to the state by a bitwise XOR operation; `SubBytes`: the non-linear byte substitution operates independently on each byte (each byte is replaced with another according to a lookup table); `ShiftRows`: it is a permutation on the 16 bytes where each row of the state is shifted cyclically a certain number of times; `MixColumns`: this operation treats each column of the state as a 4-byte vector and multiplies it by a matrix.

The high-level execution of the AES-128 algorithm is to chain the operations as follows. 1/ Key Expansion: round keys $K_0, ..., K_{10}$ are derived from the cipher key using Rijndael's key schedule; 2/ `AddRoundKey`(K_0);
3/ for i from 1 to 9 do `SubBytes`- `ShiftRows`- `MixColumns`- `AddRoundKey`(K_i) ; And finally 4/ `SubBytes`- `ShiftRows`- `AddRoundKey`(K_{10}).

3.2 Our Proposal

Let k be the number of bits in the secret data handled in a cryptographic algorithm. The objective is to encode the k data bits into $n > k$ bits, such that an adversary who observes a bit subset of size $\mu < n$ can gain no information on the secret data. We choose a code C with a generator matrix G and a parity matrix H, a set $L = (l_1, ..., l_k)$ of k linearly independent vectors from $\{0,1\}^n \backslash C$ and we encode a vector $x = (x_1, ..., x_k) \in \{0,1\}^k$ as explained in Section 2: $z = x.L \oplus c$ where $x.L = \sum_{i=1}^{k} x_i l_i$ (modulo 2) and $c = m.G$ is a random codeword.

The code C is constructed in advance by selecting its parity matrix H as the generator matrix of an $[n, k, \mu + 1]$ binary linear code and L can be computed later. The k linearly independent n-bit vectors $l_1, ..., l_k$ can be fixed for each target component. For example for smart cards, the set $L = (l_1, ..., l_k)$ can be separately computed for each card during the personalization step.

Remark 1. In the following, we will consider $k = 8$, i.e. that the wire-tap masking is applied at the byte level (fitting to input's size for AES S-boxes). When transforming the encryption algorithm to operate on masked data, the main overhead comes from the definition of new S-boxes compatible with such wire-tap representation. As for classical additive masking, for efficiency reasons we may prefer to use the same mask for all bytes (and all rounds). We consider this option in the following description of the masked algorithm. Nevertheless, if memory size is not a concern we can easily use different codewords for each byte and each round.

Pre-Configuration. In the context of the wire-tap code masking, we first define new operations `AddRoundKey`$'$, `ShiftRows`$'$ and `MixColumns`$'$ from the original linear operations `AddRoundKey`, `ShiftRows` and `MixColumns` such that they are compatible with the wire-tap representation $x.L \oplus c$. `AddRoundKey`$'$ has to expand the key by wire-tap encoding before the exclusive-OR operation. `ShiftRows`$'$ and `MixColumns`$'$ are modified such that they operate linearly on a larger state. They are defin-ed in the following way:
-`AddRoundKey`$'(K_i)(x.L) = \big(\texttt{AddRoundKey}(K_i)(x)\big).L = (x \oplus K_i).L$, for all $x \in \{0,1\}^8$, for all $i \in \{0, \ldots, 10\}$;
-`ShiftRows`$'(x^4.L^4) = \texttt{ShiftRows}(x^4).L^4$ for all $x^4 = (x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}) \in \{0,1\}^{4\times 8}$;

-`MixColumns`$'(x^4.L^4) =$ `MixColumns`$(x^4).L^4$, for all $x^4 \in \{0,1\}^{4\times 8}$.
where `AddRoundKey`$'(K_i)$ is basically the XOR between its input and $K_i.L$ and `ShiftRows`$'$, `MixColumns`$'$ are defined to be linear operations over 4 bytes (`ShiftRows` operates row per row of the AES state and `MixColumns` column per column of the AES state, the state being a 4×4 bytes array). Above, $L^4 = (L^{(1)}, L^{(2)}, L^{(3)}, L^{(4)})$ corresponds to the concatenation of four vectors of $k = 8$ linearly independent n-bit vectors (either four different or four times the same L) and $x^4.L^4$ is $x^{(1)}.L^{(1)} + x^{(2)}.L^{(2)} + x^{(3)}.L^{(3)} + x^{(4)}.L^{(4)}$.

These calculations are easy to perform because the operations are linear and done only once after the generation of L (that is set for once or renewed after some number of uses; and thus inexpensive in performance with respect to execution of an encryption). They can be done by both software and hardware implementations.

Preliminary Step. The preliminary step below is computed before encryption. Let c, c' be two random codewords from the code C, then we define a new S-box `SubBytes`$'$ as: `SubBytes`$'(x.L \oplus c) =$ `SubBytes`$(x).L \oplus c'$

The look-up table associated to `SubBytes`$'$ is then defined over $\{0,1\}^n$. This leads in general to a size of $2^n \times n$ bits. Here, as only 2^k entries are possible, we can manage to store it on $2^k \times n \times 2$ by representing it as a list of 2^k couples of n-bits input/output vectors; the factor of expansion becomes $2n/k$. For instance, with $n = 12$ (respectively $n = 16$), the required memory for one transformed S-box is 768 bytes (resp. 1024 bytes); this corresponds to an expansion factor of 3 (resp. 4). For efficiency reasons, we use the same codewords, i.e. the same `SubBytes`$'$, for all rounds of the algorithm. Nevertheless, independent choices could be envisaged if the memory size is sufficient. This requires at most $2^k \times n \times 320$ bits.

By default, new codewords are drawn before each encryption. Note however that due to the secrecy property of the wire-tap codes, you could use several times the same codewords if you can ensure that the overall number of leaked bits remains lower than the security bound μ.

Transformation of the Encryption Algorithm. We explain below the new version of the algorithm when protected by wire-tap encoding. This corresponds to the operations that are executed at each encryption. Let x the input message to be encrypted. We have the following steps (to simplify the description of the masking we consider only one byte for x, whereas of course the whole state is to be masked):

– Take c and c' at random and define `SubBytes`$'$ as above.
– Draw another random codeword c_1 and let $c_2 \leftarrow c \oplus c_1$.
– Compute $z \leftarrow x.L \oplus c_1$
– Compute $z \leftarrow$ `AddRoundKey`$'(K_0)(z) \oplus c_2$
  (i.e. $z =$ `AddRoundKey`$'(K_0)(x.L \oplus c_1) \oplus c_2 = (K_0 \oplus x).L \oplus c)$
– for i from 1 to 9
  • $z \leftarrow$ `SubBytes`$'(z)$
    (this gives $z =$ `SubBytes`$(K_0 \oplus x).L \oplus c$')

  - $z \leftarrow \texttt{ShiftRows}'(z)$
    (it leads to $z = \texttt{ShiftRows} \circ \texttt{SubBytes}(K_0 \oplus x).L \oplus \texttt{ShiftRows}'(c'))$
  - $z \leftarrow \texttt{MixColumns}'(z)$
    $(z = \texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes}(K_0 \oplus x).L \oplus \texttt{MixColumns}' \circ \texttt{ShiftRows}'(c'))$
  - $z \leftarrow \texttt{AddRoundKey}'(K_i)(z) \oplus c_2 = z \oplus K_i.L \oplus c_2$
  - $z \leftarrow z \oplus c_1 \oplus \texttt{MixColumns}' \circ \texttt{ShiftRows}'(c')$
    i.e. $z = (K_i \oplus$
    $\texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes}(K_0 \oplus x)).L \oplus c$
- $\texttt{SubBytes}'$
- $\texttt{ShiftRows}'$
- $\texttt{AddRoundKey}'(K_{10}) \oplus c_3 \oplus \texttt{ShiftRows}'(c')$ with some random codeword c_3 (this leads to $y.L \oplus c_3$ where y is the output value of the encryption via a non-masked AES implementation)
- Apply the parity matrix H and invert $H(y.L)^T$ to obtain the final result y

Note that, above, the input message, the intermediate messages and the round keys as well are always masked by some codewords when manipulated.

4 Properties and Extensions

4.1 Simple Power Analysis.

The consumption of the new S-boxes $\texttt{SubBytes}'$ now depends on inputs/outputs encoded via the wire-tap codes. Thus by definition, if an adversary recovers less than μ bits through one or several observations of the encryption through the same choices of codewords c and c', then by Lemma 1 he cannot gain information on the original inputs/outputs of `SubBytes`. Moreover, as the length n (and consequently μ) can be chosen large at the time of the set-up of the implementation, it is theoretically possible – assuming that the number of bits that can be read out by an adversary is bounded by a constant independent of n – to thwart side-channel analysis with large number of leaked bits.

4.2 First Order DPA.

Similarly to classical additive masking, the resistance against Differential Power Analysis at the first order is achieved thanks to the masking of the values $x.L$. The mask is (or derived from) a random codeword which is not a random vector from $\{0,1\}^n$. But as illustrated by our experiments, this difference affects the security only when the size $n-k$ of the code C (thus the number of possible codewords) is too small. When $n-k$ increases, the difficulty increases as well and this enables to obtain parameters which achieve resistance against first order DPA that is almost the same as the resistance obtained with classical additive masking technique.

4.3 Higher Order DPA.

In our construction, the length of the encoded bytes can be increased. This is an important difference with classical masking technique and this is an interesting feature when studying second-order DPA. The results of our experiment confirm that the efficiency of the second-order side-channel attack decreases when the size n increases.

Moreover, an additional way to achieve implementations that are resistant to second order DPA in practice is to keep the vector L unknown. This is somehow close to affine masking techniques from [3, 14]: the specific structure of wiretap codes – while enabling new properties that are discussed further below – still enables us to achieve in practice second-order resistance.

4.4 Unmasking without Masks.

An additional feature that comes with our technique is the possibility to unmask without the knowledge of the final codeword used. This is due to the parity matrix H that cancels out all codewords. It avoids to manipulate the mask at the last step of the algorithm that is usually a phase where an adversary could try to capture the mask when retrieved from the memory.

A usual scenario for a DPA attack is when the adversary knows the ciphertexts and wants to attack the last rounds. For instance, one can imagine a classical masking scheme. In this case, [13] considers two relevant points in time which occur during a small period of time and which are dependent on the masked value and its mask:

– The adversary measures the effect of the masked operations during the last rounds of the encryption.
– At the end of the AES encryption, the adversary measures the consumption when the random mask is retrieved to unmask the result.

To mount its HO-DPA attack, the adversary has to combine the elements coming from these two points in time.

In this context, the possibility to unmask without the use of the mask is clearly an asset.

To increase the difficulty, we can moreover choose to pre-compute the masking version of the final round key $(K_{10}.L \oplus c_3 \oplus \texttt{ShiftRows}'(c'))$ at some random place during the encryption.

For specific choices of C and L, this possibility can be generalized to all unmasking steps by using the parity matrix H_1 (respectively H_2) of the code obtained as $\texttt{MixColumns}'(\texttt{ShiftRows}'(C))$ (resp. $\texttt{ShiftRows}'(C)$), and choosing L not in these codes either.

4.5 Fault Detection.

Finally we can use the wire-tap encoding of a message to add a kind of fault detection step by exploiting the fact that an encoded message x has a special format ($z = x.L \oplus c$): Apply H to z^T to obtain $H(x.L)^T$, invert the result to recover x, compute $z' = x.L \oplus c$ and check whether $z = z'$.

References

1. Akkar, M.L., Giraud, C.: An implementation of DES and AES, secure against some attacks. In: Çetin Kaya Koç, Naccache, D., Paar, C. (eds.) CHES. Lecture Notes in Computer Science, vol. 2162, pp. 309–318. Springer (2001)
2. Blömer, J., Guajardo, J., Krummel, V.: Provably secure masking of aes. In: Handschuh, H., Hasan, M.A. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 3357, pp. 69–83. Springer (2004)
3. Fumaroli, G., Martinelli, A., Prouff, E., Rivain, M.: Affine masking against higher-order side channel analysis. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 6544, pp. 262–280. Springer (2010)
4. Li, Y., Sakiyama, K., ichi Kawamura, S., Komano, Y., Ohta, K.: Security evaluation of a dpa-resistant s-box based on the fourier transform. In: Qing, S., Mitchell, C.J., Wang, G. (eds.) ICICS. Lecture Notes in Computer Science, vol. 5927, pp. 3–16. Springer (2009)
5. MacWilliams, F.J., Sloane, N.J.A.: The theory of error correcting codes. North-Holland (1977)
6. National Institute of Standards and Technology: Advanced Encryption Standard (FIPS PUB 197) (November 2001), `http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf`
7. Oswald, E., Mangard, S., Pramstaller, N.: Secure and efficient masking of aes - a mission impossible? Cryptology ePrint Archive, Report 2004/134 (2004)
8. Ozarow, L.H., Wyner, A.D.: Wire–tap channel II. Bell Systems Technical Journal 63(10), 2135–2157 (1984)
9. Ozarow, L.H., Wyner, A.D.: Wire-tap channel ii. In: EUROCRYPT. pp. 33–50 (1984)
10. Prouff, E., Giraud, C., Aumônier, S.: Provably secure s-box implementation based on fourier transform. In: Goubin, L., Matsui, M. (eds.) CHES. Lecture Notes in Computer Science, vol. 4249, pp. 216–230. Springer (2006)
11. Schramm, K., Paar, C.: Higher order masking of the AES. In: Pointcheval, D. (ed.) CT-RSA. Lecture Notes in Computer Science, vol. 3860, pp. 208–225. Springer (2006)
12. Thangaraj, A., Dihidar, S., Calderbank, A.R., McLaughlin, S.W., Merolla, J.M.: Capacity achieving codes for the wire tap channel with applications to quantum key distribution. CoRR cs.IT/0411003 (2004)
13. Tillich, S., Herbst, C.: Attacking state-of-the-art software countermeasures-a case study for aes. In: Oswald, E., Rohatgi, P. (eds.) CHES. Lecture Notes in Computer Science, vol. 5154, pp. 228–243. Springer (2008)
14. von Willich, M.: A technique with an information-theoretic basis for protecting secret data from differential power attacks. In: Honary, B. (ed.) IMA Int. Conf. Lecture Notes in Computer Science, vol. 2260, pp. 44–62. Springer (2001)
15. Wyner, A.D.: The Wire-tap Channel. Bell Systems Technical Journal 54(8), 1355–1387 (January 1975)

Author Index