Multicast Authentication in Fully Adversarial Networks

Anna Lysyanskaya Roberto Tamassia Nikos Triandopoulos
Brown University
Department of Computer Science
Providence, RI, 02912-1910
{anna,rt,nikos}@cs.brown.edu

Abstract

We study a general version of the multicast authentication problem where the underlying network, controlled by an adversary, may drop chosen packets, rearrange the order of the packets in an arbitrary way, and inject new packets into the transmitted stream. Prior work on the problem has focused on less general models, where random, rather than adversarially-selected, packets may be dropped and altered, or no additional packets may be injected into the stream. We describe an efficient and scalable authentication scheme that is based on a novel combination of error-correcting codes with standard cryptographic primitives. We prove the security of our scheme and analyze its performance in terms of the computational effort at the sender and receiver and the communication overhead. We also discuss specific design and implementation choices and compare our scheme with previously proposed approaches.

1. Introduction

The authentication of multicast transmissions of data streams over the Internet is a challenging problem. IP multicast is implemented with a best-effort delivery mechanism over the UDP transport protocol, where packet losses are tolerated. Thus, the received stream may differ from the transmitted one. Any authentication scheme for multicast streams should verify as many as possible of the received packets without assuming the availability of the entire original stream. In addition, it should resist against any types of attacks by an adversary, even when the adversary controls the underlying network.

In the *multicast authentication problem*, we wish to authenticate a packet stream transmitted over a network that may adversarially drop packets, arbitrarily rearrange the order of the packets, and inject new packets into the stream. Prior work on the subject has focused on a network model where either all the received packets are valid (authentic) or packets are lost according to some predefined random patterns (e.g., [12, 19, 22, 27]) or no packet injections occur (e.g., [20, 21]). Thus, most of the previously proposed schemes only tolerate *erroneous* network behavior and are not resilient against an *adversarial* behavior of the network.

Of course, if each packet were signed by the sender, then the only damage the adversarial network could inflict is packet loss, as the receiver would simply reject packets whose signature is not verified. However, this simple "sign-all" solution is undesirable because of the repeated use by the sender of the critical and computationally expensive sign primitive for each transmitted packet and the heavy communication overhead caused by the addition of a signature to each packet. Additionally, this solution suffers by a simple denial-of-service attack at the receiver; one signature verification must be performed for each received packet, valid or not.

In this paper, we formally define a general model for multicast authentication where an adversary can perform various attacks on the transmitted streams. In this model, two parameters of the network, the *survival rate* and the *flood rate*, characterize the power of the adversary. We describe an efficient authentication scheme for this model that gives almost the same security guarantees as if each packet were individually signed, but requires only one signature operation for the entire stream and adds only a constant size authentication overhead per packet. Our technique uses a novel combination of Reed-Solomon error-correcting codes with standard cryptographic primitives, such as collision-resistant hashing and digital signatures.

In the rest of this section, we introduce our model, summarize our contributions and review previous work on multicast authentication. The cryptographic primitives and error-correcting codes used in this paper are reviewed in Section 2. In Section 3, we describe in detail our adversarial network model and multicast authentication framework. Section 4 describes our multicast authentication scheme and gives proofs of correctness and security. In Section 5 we analyze the performance of our scheme and compare it with various other proposed schemes in terms of security assumptions, underlying network model, resilience to packet loss and injection, computational effort at the sender and receiver, and communication overhead. Conclusions and future work are given in Section 6.

1.1. Model and Contributions

We consider the problem of authenticating a stream of packets transmitted over a *fully adversarial* network. Namely, the network is controlled by an adversary who can destroy packets of her choice, arbitrarily rearrange the order of the packets, and inject new, arbitrarily constructed, packets. We limit the power of the adversary to modify a stream of n packets transmitted by the sender by introducing two parameters of the network, the *survival rate* α, $0 < \alpha \leq 1$, and the *flood rate* β, $\beta \geq 1$, which are assumed to be constants. A network with these two parameters, which we call an (α, β)*-network*, guarantees that despite the presence of the adversary, at least αn packets in the received stream are valid and the received stream contains at most βn packets. The model is formally described in Section 3. For now, we justify the introduction of the survival and flood rates with the following observations. If too many packets are dropped or corrupted by the adversary, then the main problem is the loss of data, as the small number of valid packets received may be useless even if authenticated. On the other hand, if the adversary can inject a very large number of packets, then we have a denial-of-service attack.

The contributions of our work can be summarized as follows:

- We provide a formal definition of multicast authentication over an (α, β)-network, where arbitrary packets are lost, injected, and rearranged, subject to a given survival rate α and flood rate β. We also give the requirements for an authentication scheme to be *correct* and *secure*.

- We present the first efficient and scalable multicast authentication scheme for an (α, β)-network. Our scheme is based on digital signatures, cryptographic hash functions and Reed-Solomon error-correcting codes. This last feature of our scheme provides a new interesting connection between coding theory and security.

- We prove the correctness and security of our scheme, analyze its performance in terms of various cost parameters, discuss design and implementation choices, and compare it with previous approaches. In particular, we show that our scheme adds to each transmitted packet only a small amount of authentication information, proportional to β/α^2, and that all the valid packets received are recognized, while all the invalid packets are rejected.

The only prior approach that provides security in our adversarial model and is the inefficient "sign every packet" solution, which consists of either (i) singing each packet individually or (ii) using a Merkle hash tree [28]. The trivial solution of signing each packet individually is not viable due to heavy computational operations at both the sender and the receiver, but also because secret-key operations are expensive in terms of the security architecture as well. The Merkle-tree-based authentication scheme has the drawback that the communication overhead (signature and hash values) grows with the number of packets sent.

1.2. Prior and Related Work

Previous work on multicast authentication considers both unconditionally secure and computationally secure authentication. Approaches based on the information theoretic model (see, e.g., [8, 26]) tend to be less practical. In the rest of this section, we overview approaches that use computationally secure authentication.

MAC-Based Approaches. Various approaches use message authentication codes (MACs) and secret-key cryptography. The trivial solution here is having the group members sharing a secret key and including a MAC into every packet sent, but this scheme is not secure, as any user can spoof packets. In another MAC-based trivial solution, each receiver has her own secret key and the sender possesses all such keys. To authenticate a stream, the sender adds to each packet a MAC for every receiver. This approach is not scalable because of the high communication cost. In [4], a MAC-based scheme is described that is secure with high probability against any coalition of w corrupted users and where $O(w)$ MACs are appended to each packet. This scheme is not fully scalable due to its communication overhead. In [22], another scheme that uses MACs is proposed, where a MAC is appended to every packet and the key of the MAC is provided in some subsequent packet. To tolerate packet losses, the keys are generated by means of a hash chain. This approach has low communication overhead. However, it requires time synchronization between the parties. Two MAC-based schemes that make explicit use of the topology of a multicast tree are presented in [29]. Both schemes are similar in concept to [22] and take denial-of-service and access control into consideration (namely, a corrupted packet is filtered out as soon as possible in the multicast tree and only legitimate group subscribers can authenticate the multicast packets). Both schemes assume the existence of secure and trusted routers at the nodes of the tree. In addition, the first scheme uses clock synchronization, whereas the second scheme relies on the existence of secure channels between the source and each of the receivers.

Boneh et al. [1] generalize MACs to a multicast setting by defining a new primitive for multicast authentication called multicast MAC (MMAC). They show that any MMAC scheme can be transformed into a digital signature scheme of almost the same efficiency. Thus, any multicast authentication scheme not relying on additional assumptions on the network (such as synchronization, trusted routers, or secure channels) may as well use a signature scheme! This brings us to *signature amortization*. Other research efforts have focused on building faster signature schemes for signing every packet separately (e.g., [10, 25]).

Signature Amortization. Other approaches use the mechanism of signature amortization, where a single digital signature is used for the authentication of multiple packets. A first scheme that uses signature amortization over a hash chain appears in [10]. Each packet p_i is augmented with authentication information a_i, which is recursively defined as the hash of $p_{i+1} \circ a_{i+1}$ ($\circ$ denotes concatenation). Also, the augmented first packet $p_1 \circ a_1$ is digitally signed. This scheme has constant authentication overhead per packet but does not tolerate packets losses. In [28], a Merkle hash tree is used to amortize a signature over n packets. Namely, a hash tree is built on top of the hashes of the packets and the root hash value is digitally signed. Each packet is augmented with authentication information that consists of the signed root hash and the hashes of the siblings of the nodes on the path between the root and the leaf associated with the packet. The scheme tolerates packet losses but has logarithmic communication overhead per packet. In contrast, our approach, which also uses signature amortization, has *constant* communication overhead.

Graph-Based Authentication. Graph-based authentication [12, 19, 22, 27] generalizes the idea of amortizing a signature over a hash chain in such a way as to tolerate packet losses. A single-sink directed acyclic graph (DAG) G is defined, where each vertex corresponds to a packet. A directed edge from packet p_i to packet p_j indicates that the authentication information a_j of packet p_j includes the hash of $p_i \circ a_i$. Also, the augmented packet $p_1 \circ a_1$ of the sink of the DAG is digitally signed. The validation of packets proceeds backward along the edges of the graph. Namely, if packet p_j has been validated and edge (p_i, p_j) exists in G, then the validity of packet p_i can be determined using the authentication information a_j of p_j. Graph-based authentication schemes offer probabilistic security guarantees provided packet losses occur randomly (i.e., they are not adversarially selected). In particular, they require that the signature packet will reach the receiver intact. Two packet loss patterns have been studied: the uniform model, where each packet is lost with a fixed probability and independently of other packets being lost, and the bursty model, where a packet is lost with a fixed probability and then a given number of successive packets are also lost.

In [22], G is an augmented-chain graph, consisting of a path plus additional edges that connect vertices at various distances. In [12], another augmented-chain graph is designed specifically to tolerate bursty packet losses. Random graphs and a new scheme that is resilient to multiple bursty losses are studied in [19]. Finally, in [27], expander graphs are used. The efficiency of graph-based authentication schemes is analyzed in [5] and experimentally studied in [7].

Erasure Codes. In [21] and [20], erasure codes (e.g., [16, 17, 23]) are used for multicast authentication to tolerate adversarially-chosen packet losses and disperse one signature over the packets in a group. The constructions are efficient in terms of communication cost and similar in principle. The two schemes only differ in that in [20], encoding is performed twice to reduce the size of the authentication information. Both schemes are, however, vulnerable to a very simple attack: a single injected packet can compromise the correctness of the decoding procedure at the receiver. In [15], a "binding" of the valid packets through the use of a Merkle hash tree has been recently proposed to tolerate packet injections: each packet carries also the Merkle-tree authentication information so that blocks of valid packets are grouped together and blocks of invalid packets are filtered out. This scheme suffers from high, *not constant*, per-packet communication overhead.

2. Preliminaries

In this section, we introduce some notation (mostly from [18]) and define the cryptographic and coding primitives that we use in our construction.

2.1. Notation

Let A be an algorithm. By $A(\cdot)$ we denote that A has one input (resp., by $A(\cdot, \ldots, \cdot)$ we denote that A has several inputs). By $y \leftarrow A(x)$, we denote that y was obtained by running A on input x. If A is deterministic, then this y is unique; if A is probabilistic, then y is a random variable. If S is a finite set, then $y \leftarrow S$ denotes that y was chosen from S uniformly at random. By $y \in A(x)$ we mean that the probability that y is output by $A(x)$ is positive.

By $A^O(\cdot)$, we denote an algorithm that makes queries to an oracle O. I.e., this algorithm (Turing machine) will have an additional (read/write-once) query tape, on which it will write its queries in binary; once it is done writing a query, it inserts a special symbol "#". By external means, once the symbol "#" appears on the query tape, oracle O is invoked and its answer appears on the query tape adjacent to the "#"

symbol. By $Q = Q(A^O(x)) \leftarrow A^O(x)$ we denote the contents of the query tape once A terminates, with oracle O and input x. By $(q, a) \in Q$ we denote the event that q was a query issued by A, and a was the answer received from oracle O.

Let b be a boolean function. By $(y \leftarrow A(x) : b(y))$, we denote the event that $b(y)$ is TRUE after y was generated by running A on input x. The statement $\Pr[\{x_i \leftarrow A_i(y_i)\}_{1\leq i\leq n} : b(x_n)] = \alpha$ means that the probability that $b(x_n)$ is TRUE after the value x_n was obtained by running algorithms $A_1, \ldots, A_n$ on inputs $y_1, \ldots, y_n$, is α, where the probability is over the random choices of the probabilistic algorithms involved.

2.2. Cryptographic Primitives

The following definition is due to Goldwasser, Micali, and Rivest [11], and has become the standard definition of security for signature schemes. Schemes that satisfy it are also known as signature schemes secure against *adaptive chosen-message attack*.

Definition 2.1 (Signature scheme). *Probabilistic polynomial-time algorithms* $(G(\cdot), \mathsf{Sign}_{(\cdot)}(\cdot), \mathsf{Verify}_{(\cdot)}(\cdot,\cdot))$, *where* G *is the key generation algorithm,* Sign *is the signature algorithm, and* Verify *the verification algorithm, constitute a digital signature scheme for a family (indexed by the public key* PK*) of message spaces* $\mathcal{M}_{(\cdot)}$ *if the following two hold:*

Correctness. If a message m *is in the message space for a given public key* PK*, and* SK *is the corresponding secret key, then the output of* $\mathsf{Sign}_{SK}(m)$ *will always be accepted by the verification algorithm* Verify_{PK}*. More formally, for all values* m *and* k*:*

$$\Pr[(PK, SK) \leftarrow G(1^k); \sigma \leftarrow \mathsf{Sign}_{SK}(m) : m \leftarrow \mathcal{M}_{PK} \wedge \neg\mathsf{Verify}_{PK}(m, \sigma)] = 0.$$

Security. Even if an adversary has oracle access to the signing algorithm that provides signatures on messages of the adversary's choice, the adversary cannot create a valid signature on a message not explicitly queried. More formally, for all families of probabilistic polynomial-time oracle Turing machines $\{A_k^{(\cdot)}\}$*, there exists a negligible function*[1] $\nu(k)$ *such that*

$$\Pr[(PK, SK) \leftarrow G(1^k); (Q, m, \sigma) \leftarrow A_k^{\mathsf{Sign}_{SK}(\cdot)}(1^k) : \mathsf{Verify}_{PK}(m, \sigma) = 1 \wedge \neg(\exists\sigma' \mid (m, \sigma') \in Q)] = \nu(k).$$

For completeness, we give a standard definition of a family of collision-resistant hash functions.

1 A function $\nu : \mathbb{N} \rightarrow \mathbb{R}$ is negligible if for every positive polynomial $p(\cdot)$ and for suffi ciently large k, $\nu(k) < \frac{1}{p(k)}$.

Definition 2.2 (Collision-resistant Hash Function). *Let* H *be a probabilistic polynomial-time algorithm that, on input* 1^k*, outputs an algorithm* $\mathcal{H} : \{0,1\}^* \mapsto \{0,1\}^k$*. Then* H *defines a family of collision-resistant hash functions if:*

Efficiency. For all $\mathcal{H} \in H(1^k)$*, for all* $x \in \{0,1\}^*$*, it takes polynomial time in* $k + |x|$ *to compute* $\mathcal{H}(x)$*.*

Collision-resistance. For all families of probabilistic polynomial-time Turing machines $\{A_k\}$*, there exists a negligible function* $\nu(k)$ *such that*

$$\Pr[\mathcal{H} \leftarrow H(1^k); (x_1, x_2) \leftarrow A_k(\mathcal{H}) : x_1 \neq x_2 \wedge \mathcal{H}(x_1) = \mathcal{H}(x_2)] = \nu(k).$$

2.3. Error Correcting Codes

Error correcting codes allow recovering a message that is transmitted over a noisy channel. Let $q \geq 2$ be the size of alphabet $[q] = \{1, 2, ..., q\}$. An *error correcting code* $[n, k]_q$ is a function $C : [q]^k \rightarrow [q]^n$, that processes a k-length message x over $[q]$ and adds redundancy to form a longer n-length codeword $C(x)$. The processing and the added redundancy help correcting up to e errors of the codeword $C(x)$; that is, given a received word $y \in [q]^n$ such that y and $C(x)$ differ in at most e characters of the alphabet, we can unambiguously correct (decode) y to $C(x)$. The value e depends on the code. If only redundancy is added by a code C, i.e., if all symbols of x appear also among the symbols of $E(x)$, then the code is *systematic*. *List-decoding* allows to ambiguously correct even more errors. Given that $C(x)$ is received as word $y \in [q]^n$, list-decoding provides a list of candidate messages, such that x belongs in this list.

Reed-Solomon codes [24] are a family of codes that are based on properties of univariate polynomials over finite fields. We next define the $[n, k+1]_q$ Reed-Solomon code, where n, k and q are positive integers and parameters of the code. We present here a slightly modified definition of Reed-Solomon codes than the one commonly used in the literature, so that, by definition, Reed-Solomon codes considered in this paper are systematic. This property is not necessary for the correctness of our scheme, but offers an extra desired property in our construction, discussed in Section 5 and requires no extra computational effort at the encoder.

Definition 2.3 (Reed-Solomon code). *A* $[n, k+1]_q$ *Reed-Solomon code consists of the following components:*

Alphabet. The alphabet is a finite field $\mathbb{F}_q$ *of size* $q \geq n$*, with* q *being a prime power.*

Encoder. The code is a function $C : \mathbb{F}_q^{k+1} \rightarrow \mathbb{F}_q^n$*, where* $n > k+1$*. In more detail, the encoder:*

1. *takes as input the parameters* n *and* k*, and* $k+1$ *points* (i, y_i)*,* $i \in \mathbb{F}_q$*,* $y_i \in \mathbb{F}_q$*,* $1 \leq i \leq k+1$*,*

2. *finds a unique univariate polynomial $p \in \mathbb{F}_q[x]$ over elements of $\mathbb{F}_q$ and of degree at most k, such that $p(i) = y_i$, $1 \leq i \leq k+1$, and*
3. *outputs points $(i, p(i))$, for $1 \leq i \leq n$. The ratio $\frac{k+1}{n} < 1$ is called the* rate *of the code.*

We have that C is a systematic code.

Decoder (GS-Decoder). Let $\epsilon > 0$ be a parameter that controls the performance of the decoder. The decoder takes as input parameters n and k, the number of errors e that may occur, and n points (x_i, y_i), $1 \leq i \leq n$, and list-decodes, i.e., it outputs a list of all univariate polynomials $p \in \mathbb{F}_q[x]$ of degree at most k such that $y_i \neq p(x_i)$ for less than e values of i, $1 \leq i \leq n$.

For a $[n, k+1]_q$ Reed-Solomon code, we use a decoder that runs in polynomial time and is due to Guruswami and Sudan [13, 14]. We refer to this decoder as GS-Decoder.

Theorem 2.1 (Guruswami-Sudan). *Consider a $[n, k+1]_q$ Reed-Solomon code. For any $\epsilon > 0$, given n points with $e = n - \sqrt{(1+\epsilon)kn}$ errors, GS-Decoder outputs a list of size $O(\epsilon^{-1}\sqrt{n/k})$ in $O(n^2\epsilon^{-5}\log^2 q \log^{O(1)} \log q)$ time, performing $O(n^2\epsilon^{-5}\log q)$ field operations.*

We view ϵ as a parameter of the GS-Decoder, $\epsilon > 0$, and use $\mathsf{GSDecode}_\epsilon(n, k, e, \{(x_i, y_i) | 1 \leq i \leq n\})$ to denote that GS-Decoder runs with parameter ϵ having as input parameters the integers n, k, e and the points (x_i, y_i), $1 \leq i \leq n$.

In practice, codes with constant expansion are used, where $k = \rho n$ for some constant $\rho < 1$. In particular, our construction makes use of a $[n, \rho n + 1]_q$ Reed-Solomon code, with $\rho < 1$, over some large alphabet of size $q = 2^c$ for some constant c. From Theorem 2.1 we have the following.

Corollary 2.2. *For any $[n, \rho n+1]_q$ Reed-Solomon code, for any constants $\epsilon > 0$ and $\rho < 1$ such that $\sqrt{(1+\epsilon)\rho} < 1$, on an input with $e = (1 - \sqrt{(1+\epsilon)\rho})n$ errors, GS-Decoder outputs a list of $O(1)$ size in $\tilde{O}(n^2)$ time, performing $\tilde{O}(n^2)$ field operations.*

GS-Decoder is based on an algorithm that solves the *polynomial reconstruction* problem: given k, t, and n points $\{(x_i, y_i), 1 \leq i \leq n\}$, where $x_i, y_i \in \mathbb{F}_q$, find a list that contains all univariate polynomials $p \in \mathbb{F}_q[x]$ of degree at most k such that $y_i = p(x_i)$ for at least t values of i, $1 \leq i \leq n$. Polynomial reconstruction and Reed-Solomon list-decoding are equivalent problems [13]. Namely, the following corollary is equivalent to Corollary 2.2.

Corollary 2.3. *For any constants $\epsilon > 0$ and $\rho < 1$, polynomial reconstruction on input ρn, t, and n points in $\mathbb{F}_q \times \mathbb{F}_q$ can be solved, in $\tilde{O}(n^2)$ time, provided $t \geq \sqrt{(1+\epsilon)\rho}n$, where $\tilde{O}(n^2)$ field operations are performed and the output list has $O(1)$ size.*

3. Network Model and Authentication Framework

A sender transmits a data stream, consisting of n packets marked with a *group identification tag*[2] (*GID*) to a receiver over an underlying "best-effort" network. No guarantees about the delivery of the packets exist in general. Furthermore, the network is an adversary of great, yet not unlimited, power. In our model, packets may be *adversarially lost, altered, delayed, or injected*. However, this adversary is not given complete freedom — if it were, then no messages would ever get delivered, and so our task would be hopeless.

3.1. The (α, β)-Network Model

We model the network as an *adversarial* entity, i.e., an entity that can *simultaneously* inflict *any possible type* of attack to the transmitted data stream. The repertoire of attacks consists of packet *losses*, *injections*, *alterations* and *rearrangements*. These modifications of the data stream are adversarially chosen so that the adversary can cause the loss of any selected packets. The ability to tolerate packet losses has been widely considered an important property of multicast authentication schemes [12, 19, 21, 22, 27, 28]. However, only a few previous schemes [21, 28] tolerate *adversarial losses*, i.e., the capability by the adversary to choose which packets are dropped and which survive.

Also, the adversary can inject packets of random or malicious structure into the stream. This type of network failure has not been studied as widely in the context of multicast authentication. In contrast, we develop robust techniques for dealing with it.

Finally the adversary can arbitrarily modify, delay or rearrange packets. Note that changing a packet corresponds to destroying (losing) it and injecting a new packet.

An adversarial network modelled with the above capabilities in terms of how the adversary is acting is what we call a *fully adversarial network*.

Definition 3.1 (Fully adversarial network). *A* fully adversarial network *is a network that is used for the transmission of a data stream and is* controlled *by an adversary. In particular, the adversary can:*

- *cause packets of* her choice *to be lost;*

2 Conventionally and without loss of generality, we consider data streams consisting of n packets, that is, at the sender, the data for transmission is arranged in groups of size n. The *GID* adds no new assumption about the transmitted stream. It corresponds to a means for the packets to be grouped together, which in practice can be provided by any network-layer transmission protocol in use. In our framework, the *GID* is used as an abstract quantity of constant size; in practice, it is a string of some small constant size, e.g., the size of a hash value (20 bytes for SHA-1).

- *inject packets (either* random *ones or with a* specific malicious structure*); and*
- arbitrarily *alter, delay or rearrange packets.*

It is realistic to assume that even if an adversary controls part of the network, there are still some honest routers and at least a fraction of the data packets goes through them. Thus, we expect some *reliability* from the network; namely, the network will faithfully deliver at least a constant fraction, α, of all the packets of a given stream. This assumption is also justified by the fact that if fewer than a constant fraction of the packets survive, then it is unlikely that meaningful information can be extracted from the surviving packets. Also, in modelling the ability of the adversary to maliciously inject invalid packets, we take the following into consideration: if the adversary injects packets at too high a rate, this will result in a denial-of-service attack. In this case, the receiver's primary concern is unlikely to be authentication. Thus, we assume that authentication is useful when the stream is expanded by no more than a constant factor β through adversarial packet injections.

Our two assumptions about the power of the adversary to modify a stream of n packets transmitted by the sender are expressed by two *parameters* of the adversarial network: the *survival rate* α and the *flood rate* β. In this paper, both rates are considered to be constants.

Definition 3.2 (Network parameters). *Consider a network through which a stream of n packets is transmitted by the sender.*

- *The* survival rate α, $0 < \alpha \leq 1$, *is the minimum fraction of the packets that are guaranteed to reach the receiver unmodified. I.e., at least αn packets in the received stream are valid.*
- *The* flood rate β, $\beta \geq 1$, *is the maximum factor by which the size of the received stream may exceed the size of the transmitted stream. I.e., at most βn packets are in the received stream.*

A network with the above characteristics in terms of adversarial behavior and reliability is what we call an (α, β)-*network* and is the basis for our multicast authentication framework.

Definition 3.3 ((α, β)-network). *An* (α, β)-network *is a fully adversarial network with survival rate α and flood rate β.*

3.2. Authentication Framework

We describe a new multicast authentication framework that is based on the (α, β)-network model. Our definition of a *multicast authentication scheme* essentially mimics the classical definition of security for signatures [11]. (This is not surprising since in [1] it is shown that the two problems are equivalent.) A signature scheme consists of key generation, signature, and verification algorithms (see Definition 2.1). Similarly, we have key generation, authentication, and decoding algorithms, specified below.
Key Generation: The key generation algorithm KeyGen is a probabilistic polynomial-time algorithm that takes as input the security parameter 1^k, and outputs the key pair (PK, SK). We write $(PK, SK) \leftarrow \mathsf{KeyGen}(1^k)$.

We assume that the sender knows both the public key PK and the secret key SK and that the receiver knows the public key PK. The following two algorithms, authenticator Auth and decoder Decode, are executed by the sender and the receiver respectively. The sender runs Auth to process data packets and create the authenticated packets. The receiver runs Decode to decode the received packets and recognize the valid ones.
Authenticator: The authenticator algorithm Auth takes as input:

- (SK, PK): the secret key and the public key.
- *GID*: the group identification tag of the data stream.
- n: the number of packets that need to be authenticated.
- α, β: the survival and flood rates (at least αn packets are valid and at most βn packets claim to belong to a given *GID*).
- $DP = \{p_1, \ldots, p_n\}$: the *data packets* that need to be authenticated.

The output of the authenticator algorithm is the set of *authenticated packets* $AP = \{a_1, \ldots, a_n\}$. We write: $AP \leftarrow \mathsf{Auth}(SK, PK, \mathit{GID}, n, \alpha, \beta, DP)$.
Decoder: The decoder algorithm Decode takes as input:

- PK: the public key
- *GID*: the group identification tag.
- n: the number of the original data packets.
- α, β: the survival and flood rates.
- $RP = \{r_1, \ldots, r_m\}$: the received packets.

The decoder either *rejects* the input (when less than αn of the received packets are valid, or more than $(\beta - \alpha)n$ packets are injected by the adversary), or produces the *output packets* $OP = \{p'_1, \ldots, p'_n\}$. Some of these packets may be empty — an empty output packet is denoted by $\varnothing$, and corresponds to the event that the decoder did not receive the corresponding authenticated packet. We write: $\{OP, \mathsf{reject}\} \leftarrow \mathsf{Decode}(PK, \mathit{GID}, n, \alpha, \beta, RP)$.

A signature scheme has two requirements: correctness and security. We have similar requirements for a multicast authentication scheme. A multicast authentication scheme is (α, β)-*correct* if, whenever at least αn correct authenticated packets are received among βn total packets, all and only the valid received packets will be decoded correctly, i.e., the corresponding data packets will be among the output packets. A multicast authentication scheme is *secure* if, even if the adversary is allowed to query the authenticator on any number of chosen inputs, the adversary cannot make the decoder output a non-authenticated set of packets.

Definition 3.4 (Multicast Authentication Scheme). *Probabilistic polynomial-time algorithms* (KeyGen, Auth, Decode) *constitute an* (α, β)*-correct and secure multicast authentication scheme if no probabilistic polynomial-time adversary* $\mathcal{A}$ *can win non-negligibly often in the following game:*

1. *A key pair is generated:*
$$(PK, SK) \leftarrow \mathsf{KeyGen}(1^k).$$
2. *The adversary* $\mathcal{A}$ *is given:*
   - *The public key* PK *as input.*
   - *Oracle access to the authenticator, i.e., for* $1 \leq i \leq poly(k)$*, where* $poly(\cdot)$ *is a polynomial, the adversary can specify the values* $(GID_i, n_i, \alpha_i, \beta_i, DP_i)$ *and obtain* $AP_i \leftarrow \mathsf{Auth}(SK, PK, GID_i, n_i, \alpha_i, \beta_i, DP_i)$*. However, the adversary cannot issue more than one query with the same group identification tag. That is to say, for all* $i \neq j$, $GID_i \neq GID_j$.
3. *At the end,* $\mathcal{A}$ *outputs a group identification tag, GID, the values* n, α *and* β*, and a set of packets,* RP.

The adversary wins *the game if one of the following violations occurs:*

Violation of the correctness property: *The adversary did managed to construct* RP *in such a way that even though it contains* $\alpha_i n_i$ *packets of some authenticated packet set* AP_i *for group identification tag* $GID_i = GID$*, the decoder still failed at identifying all the correct packets. Namely, the adversary wins if* all *of the following hold:*

- *For some* i*, the adversary's query* i *contained* $GID_i = GID$, $n_i = n$*, and* $\alpha_i = \alpha$*. Let* $DP_i = \{p_1, \ldots, p_n\} = DP$ *be the data packets associated with that query, and let* $AP_i = \{a_1, \ldots, a_n\} = AP$ *be the response of the authenticator.*
- *At least* αn *of the authenticated packets* $(a_1, \ldots, a_n)$ *are included in the received packets* RP*, i.e.,* $|RP \cap AP| \geq \alpha n$.
- *The number of received packets is at most* βn*, i.e.,* $|RP| \leq \beta n$.
- *For some* $1 \leq j \leq n$, p_j *is the* j*'th packet in the original set of data packets* DP*, such that the corresponding authenticated packet* a_j *was received, i.e.,* $a_j \in RP \cap AP$*, and yet was not decoded correctly. Namely, let* $(p'_1, \ldots, p'_n) \leftarrow \mathsf{Decode}(PK, GID, n, \alpha, \beta, RP)$*. For* p_j *it holds that* $p_j \neq p'_j$.

Violation of the security property: *The adversary did managed to construct* RP *in such a way that the decoder will output packets* $OP = \{p'_1, \ldots, p'_n\}$ *that were never authenticated by the authenticator algorithm for the group identification tag GID. More precisely, the adversary wins if* one *of the following happens:*

- *The authenticator was never queried with group identification tag GID and the size* n*, and yet the decoder algorithm does not reject. I.e.,* $\mathsf{reject} \neq OP = \mathsf{Decode}(PK, GID, n, \alpha, \beta, RP)$.
- *The authenticator was queried with the group identification tag GID, the values* n, α *and* β*, and the data packets* $DP = \{p_1, \ldots, p_n\}$*. However, some output packet* $p'_j \neq \varnothing$ *is different from the corresponding data packet* p_j*, where* $OP = \{p_1, \ldots, p_n\}$.

4. Construction

In this section we describe a multicast authentication scheme ($\mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Decode}$) that meets the definitions of the previous section. In the sequel, we denote with $\epsilon > 0$ the *tolerance parameter* of the decoder, which yields a trade-off between the error-tolerance ability of the decoder and its performance. By $\circ$, we denote concatenation and by $\varnothing$ we appropriately denote either a packet that is empty or the empty string. We also often omit the floor and ceiling notation in order to avoid notational overload.

4.1. Key Generation

We assume that a signature scheme ($G(\cdot)$, $\mathsf{Sign}_{(\cdot)}(\cdot)$, $\mathsf{Verify}_{(\cdot)}(\cdot, \cdot)$) and a collision-resistant hash function $\mathcal{H}$ are given (see Definitions 2.1 and 2.2). If $(PK_\sigma, SK_\sigma) \leftarrow G(1^k)$, we set $PK = (PK_\sigma, \mathcal{H})$ and $SK = SK_\sigma$.

4.2. Authenticator Auth

Input: The secret key SK, the public key PK, the group identification tag *GID*, the data stream size n, parameters α and β of the network and the data packets $DP = \{p_1, \ldots, p_n\}$.
Algorithm: For $1 \leq i \leq n$, compute the hash value $h_i = \mathcal{H}(p_i)$. The concatenation of all the hash values,

together with the value *GID*, is digitally signed: $\sigma \leftarrow \mathsf{Sign}_{SK}(\textit{GID} \circ h_1 \circ \ldots \circ h_n)$. The string $S = h_1 \circ \ldots \circ h_n \circ \sigma$ is called the *authentication information*. We want to guarantee that, even if only an α fraction of the packets survive, and a large number of packets $(\beta - \alpha)n$ are injected, the receiver still gets all the authentication information. To that end, we encode S using a Reed-Solomon code in a manner that is tolerant to packet loss and insertion; namely:

1. Let the rate of the code be $\rho = \frac{\alpha^2}{(1+\epsilon)\beta}$. Recall that α and β are the survival and flood rates of the network, respectively, whereas ϵ is the tolerance parameter of the decoder. Observe that since $\alpha \le 1$, $\beta \ge 1$ and $\epsilon > 0$, we have $\rho < 1$.
2. Split S into $\rho n + 1$ substrings of size $\left\lceil \frac{|S|}{\rho n+1} \right\rceil$, where each substring is viewed as a value of $\mathbb{F}_q$, with $q = 2^{\left\lceil \frac{|S|}{\rho n+1} \right\rceil}$ [3]. If S is not an exact multiple of $\rho n + 1$, pad S with ℓ 0's, such that $|S \circ 0^\ell| \mod \rho n + 1 \equiv 0$.
3. Treat the resulting set of $\rho n + 1$ field elements as an input to the Reed-Solomon encoder (see Definition 2.3). Compute the corresponding codeword $E(S)$ using an $[n, \rho n + 1]_q$ Reed-Solomon code[4]. $E(S)$ consists of n elements of $\mathbb{F}_q$, denoted as $(s_1, \ldots, s_n)$.
4. Let $AP = \{a_1, \ldots, a_n\}$, where for $1 \le i \le n$, we have $a_i = \textit{GID} \circ i \circ p_i \circ s_i$.

Output: Authenticated packets $AP = \{a_1, \ldots, a_n\}$.

4.3. Decoder Decode

Our decoder Decode uses a modification of the GS-Decoder (see Definition 2.3) as a subroutine. The standard GS-Decoder expects to receive, as input, n pairs (x_i, y_i), and outputs a list L of all the polynomials of degree at most k such that every $p \in L$ has the property that for at least $\sqrt{(1+\epsilon)kn}$ of the i's, $p(x_i) = y_i$. We write $L \leftarrow \mathsf{GSDecode}_\epsilon(n, k, \sqrt{(1+\epsilon)kn}, \{(x_i, y_i) | 1 \le i \le n\})$. The modified decoder is specified by parameters that are slightly different: it takes as input up to βn points (x_i, y_i) and finds a list of inputs that can be encoded by polynomials of degree at most ρn (with $\rho = \frac{\alpha^2}{(1+\epsilon)\beta}$) such that each polynomial agrees with at least αn of the input points (see Corollary 2.3). The modified decoder is obtained by adapting the original GS-Decoder, as follows:
Modified GS-Decoder $\mathsf{MGSDecoder}_\epsilon$
Input: n, α, β, and m points (x_i, y_i), $1 \le i \le m$.
Algorithm:

1. If $m > \beta n$, reject.
2. Else if there are fewer than αn distinct values of x_i, reject.
3. Else, run the GS-Decoder, that is, let $L \leftarrow \mathsf{GSDecode}_\epsilon(m, \rho n, \alpha n, \{(x_i, y_i) | 1 \le i \le m\})$, where $\rho = \frac{\alpha^2}{(1+\epsilon)\beta}$. If L is empty, reject.
4. Process $L = \{Q_1(x), \ldots, Q_\ell(x)\}$ as follows: for each $Q(i) \in L$, evaluate $Q(i)$ for $1 \le i \le \rho n + 1$ and let the string $Q_j(1) \circ Q_j(2) \circ \cdots \circ Q_j(\rho n + 1)$ be a candidate c_i.

Output: List of all computed candidates $\{c_1, \ldots, c_\ell\}$ or reject.

Lemma 4.1. $\mathsf{MGSDecoder}_\epsilon$ *runs in time* $\tilde{O}(n^2)$*, where* $\tilde{O}(n^2)$ *field operations are involved, and outputs the constant size list of all candidate inputs that are consistent with* αn *of the received points.*

Proof. All claims follow from Theorem 2.1, Corollary 2.3 and the fact that GS-Decoder operates even when the x_i's are not distinct (see Guruswami and Sudan [14]). Observe that Corollary 2.3 holds, since, if $m = \gamma n$, $\alpha \le \gamma \le \beta$, then $t \ge \alpha n \ge \sqrt{\frac{\gamma}{\beta}} \alpha n = \sqrt{(1+\epsilon)\rho n m}$. (Note that the correct input is guaranteed to be contained in the output list, since it is a polynomial of degree at most ρn that is consistent with αn points.) □

Now, we are ready to describe our decoder:
Decoder Decode_ϵ
Input: The public key PK, a group identification tag *GID*, n, parameters α and β and the received packets $RP = \{r_1, \ldots, r_m\}$.
Algorithm:

1. View packets in RP as $r_i = \textit{GID}_i \circ j_i \circ p_i \circ s_i$.
2. Discard all non-conforming packets, i.e., all packets for which $\textit{GID}_i \ne \textit{GID}$ or packets with $j_i \notin [1..n]$. Let $(r_1, \ldots, r_{m'})$ be the remaining packets in RP. Each of them is viewed as $r_i = \textit{GID} \circ j_i \circ p_i \circ s_i$, such that $j_i \in [1..n]$.
3. If $m' < \alpha n$ or $m' > \beta n$, then reject.
4. For $1 \le i \le m'$, set $(x_i, y_i) = (j_i, s_i)$
5. Run algorithm $\mathsf{MGSDecoder}_\epsilon$ with input parameters n, α, β and the m' points (x_i, y_i), $1 \le i \le m'$. If $\mathsf{MGSDecoder}_\epsilon$ rejects, reject; otherwise, obtain the candidate codewords $\{c_1, \ldots, c_\ell\}$.
6. For $1 \le i \le n$, set $h_i = \varnothing$. Let $j = 1$. While $j \le \ell$:
   - Parse the codeword c_j as string $h_1^j \circ \ldots \circ h_n^j \circ \sigma$.
   - If $\mathsf{Verify}_{PK}(\textit{GID} \circ h_1^j \circ \ldots \circ h_n^j, \sigma) = 1$, then set $h_i = h_i^j$ for $1 \le i \le n$ and break out of the loop; otherwise, increment j.

3 We see that q is in fact a function of n, α and β, thus it need not be transmitted to the receiver. Observe that $|S|$ is a function of n.

4 We assume that the value of ϵ is known also to the encoder; thus, in fact $E(S) = E_\epsilon(S)$.

7. If $(h_1, \ldots, h_n) = (\varnothing, \ldots, \varnothing)$, reject. Else, compute the output packets OP as follows:
    - Initialize $OP = \{p'_1, \ldots, p'_n\}$: for each $1 \leq i \leq n$, set $p'_i = \varnothing$.
    - For $1 \leq i \leq m'$:
        - view r_i as $r_i = \mathit{GID} \circ j \circ p_j \circ s_j$, such that $j \in [1..n]$.
        - if $\mathcal{H}(p_j) = h_j$, set $p'_j = p_j$.
8. Let $OP = \{p'_1, \ldots, p'_n\}$.

Output: $OP = \{p'_1, \ldots, p'_n\}$ or reject.

We postpone the analysis of the running time of these algorithms until the next section.

4.4. Correctness and Security Proofs

Let us show that our scheme satisfies Definition 3.4. Suppose that we have an adversary $\mathcal{A}$ who manages to break the (α, β)-correctness or security of our scheme with (non-negligible) probability $\pi(k)$. Then one of the following is true:

- With probability $\pi(k)/2$, the adversary $\mathcal{A}$ violates the (α, β) correctness property.
- With probability $\pi(k)/2$, the adversary $\mathcal{A}$ violates the security property.

Let us show that a non-negligible probability of either event contradicts the security properties of the underlying signature scheme and hash function.

Claim 4.2. *If a polynomial-time adversary $\mathcal{A}$ violates the (α, β)-correctness property of our scheme, then the underlying signature scheme is not secure, or the underlying hash function is not collision-resistant.*

Proof. Let us prove the claim by exhibiting a reduction which transforms an attack that violates the correctness of our scheme, into an attack on the underlying signature scheme.

Reduction. The input to the reduction is the public key PK_σ of the signature scheme. Our reduction is also given oracle access to the corresponding signer Sign_{SK}. The reduction sets up the public key $PK = (PK_\sigma, \mathcal{H})$. Our reduction does not know the corresponding secret key. Our reduction invokes the adversary $\mathcal{A}$ on input PK. It now needs to be able to answer the adversary's queries to the authenticator Auth. In order to respond to a query $(\mathit{GID}_i, n_i, \alpha_i, \beta_i, DP_i)$, run the algorithm Auth with the following modification: at the beginning of the algorithm Auth, instead of computing the signature σ_i, obtain it by querying the signature oracle Sign_{SK}. Everything else is carried out as prescribed by the algorithm Auth.

It is clear that the view of the adversary in this reduction will be identical to the view that the adversary obtains in real life. Therefore, with the same probability as in real life, the adversary violates the correctness property. Namely, it outputs values GID, n, α, β and the set of received packets RP, such that all of the following hold:

1. $\mathit{GID} = \mathit{GID}_i$, $n = n_i$, $\alpha = \alpha_i$, and $\beta = \beta_i$ for some i. Let $DP_i = \{p_1, \ldots, p_n\}$ be the data packets associated with that query, and let AP be the response that we gave to the adversary. In particular, let σ_i be the signature associated with this query, that is, $\sigma_i \leftarrow \mathsf{Sign}_{SK}(\mathit{GID} \circ h(p_1) \circ \ldots \circ h(p_n))$.
2. $|RP \cap AP| \geq \alpha n$ and $|RP| \leq \beta n$.
3. For some j such that $r_j \in RP$, $p_j \neq p'_j$,where $(p'_1, \ldots, p'_n) \leftarrow \mathsf{Decode}(PK, \mathit{GID}, n, \alpha, \beta, RP)$.

Case 1. Suppose that $p'_j \neq \varnothing$. From 3, we get that either $h(p_j) \neq h(p'_j)$, or it is easy to find a collision to the hash function. By definition of Decode, if $\varnothing \neq p'_j \in RP$, then, in Step 6, the algorithm Decode processes a candidate $c = h_1 \circ \ldots \circ h_n \circ \sigma$ such that $\mathsf{Verify}_{PK}(\mathit{GID} \circ h_1 \circ \ldots \circ h_n, \sigma) = 1$. We must argue that our signature oracle was never queried on input $(\mathit{GID} \circ h_1 \circ \ldots \circ h_n)$. Note that the only time it was queried with this GID, it was when we obtained σ_i on input $(\mathit{GID} \circ h(p_1) \circ \ldots \circ h(p_n))$. Moreover, in Step 7, Decode includes p'_j into OP if and only if $h(p'_j) = h_j$. Therefore, $h_j \neq h(p_j)$, and so our signature oracle was never queried with $(\mathit{GID} \circ h_1 \circ \ldots \circ h_n)$, and yet our adversary has caused us to compute a signature σ such that $\mathsf{Verify}_{PK}(\mathit{GID} \circ h_1 \circ \ldots \circ h_n, \sigma) = 1$. Thus, the underlying signature scheme is insecure.

Case 2. So, suppose that $p'_j = \varnothing$. From 1 and 2, we know that αn of the original authenticated packets were received, among the total of βn packets. Then, by the properties of $\mathsf{MGSDecoder}_\epsilon$ (Lemma 4.1), Step 5 of the algorithm Decode includes the candidate value $c = h(p_1) \circ \ldots \circ h(p_n) \circ \sigma_i$. Then, by construction, it cannot be the case that in Step 7, $(h_1, \ldots, h_n) = (\varnothing, \ldots, \varnothing)$. If $(h_1, \ldots, h_n) = (h(p_1), \ldots, h(p_n))$, then by construction of Decode, if (as is the case according to 3) $r_j \in RP$, then $p'_j \neq \varnothing$, because p'_j is set to p_j when the packet r_j is considered in Step 7. Therefore, $(h_1, \ldots, h_n) \neq (h(p_1), \ldots, h(p_n))$, and yet $\mathsf{Verify}_{PK}(\mathit{GID} \circ h_1 \circ \ldots \circ h_n, \sigma) = 1$. But the only query with GID that we ever issued to the signer was for the message $(\mathit{GID} \circ h(p_1) \circ \ldots \circ h(p_n)) \neq (\mathit{GID} \circ h_1 \circ \ldots \circ h_n)$. Thus σ is a successful forgery. $\square$

Claim 4.3. *If a polynomial-time adversary $\mathcal{A}$ violates the security property of our scheme, then the underlying signature scheme is not secure, or the underlying hash function is not collision-resistant.*

Proof. Consider setting up the reduction in exactly the same way as in the proof of Claim 4.2. Again, the adversary's view in the reduction is the same as in real life. So, just as often as in real life, the adversary will violate the security property of our scheme, namely, one of the following will hold:

1. The authenticator was never queried with group identification tag *GID* and size n, and yet the decoder algorithm does not reject. I.e., $\mathsf{reject} \neq OP = \mathsf{Decode}(PK, \mathit{GID}, n, \alpha, \beta, RP)$.
2. The authenticator was queried with the group identification tag *GID*, with the values n, α and β, and data packets $DP = \{p_1, \ldots, p_n\}$. However, some output packet $p'_j \neq \varnothing$ is different from the corresponding data packet p_j, where $OP = \{p'_1, \ldots, p'_n\}$.

Suppose 1 holds. Then, from the description of the decoder, we know that the only way that it will produce some non-empty set of output packets is if, in Step 6, it sees a string c and a signature σ such that $\mathsf{Verify}_{PK}(\mathit{GID} \circ c, \sigma) = 1$. Since the signature oracle was never queried for this *GID* and n, σ is a successful forgery.

So, suppose that 2 holds. Then this is exactly the same situation as Case 1 of the proof of Claim 4.2, and we obtain either a successful forgery or a hash function collision in the same manner. $\square$

5. Analysis

We now analyze our scheme in terms of the various cost parameters. Recall that: α ($0 < \alpha \leq 1$) is the survival rate of the network, β ($\beta \geq 1$) is the flood rate of the network, ϵ ($\epsilon > 0$) is the tolerance parameter of the list-decoder, and $\rho + \frac{1}{n}$ is the rate of the encoder, where

$$\rho = \frac{\alpha^2}{(1+\epsilon)\beta} < 1.$$

In the sequel, by h we denote the size of a hash value and by s the size of a digital signature.

Computational Cost. The sender and the receiver execute algorithms Auth and Decode, respectively. Both algorithms involve field operations (additions and multiplications) over finite field $\mathbb{F}_q$ of size $q = 2^{\lceil \frac{nh+s}{\rho n+1} \rceil} \simeq 2^{\frac{h}{\rho}}$. Both operations take $O\left(\frac{h}{\rho} \log^{O(1)} \frac{h}{\rho}\right)$ time [13]. Setting $N = \frac{h}{\rho}$, both operations take $O(N \log^{O(1)} N)$ time. Note that N is independent of n.

Authenticator: The cost to encode n packets is as follows. First, n hashes are computed and one signature operation is performed over the hashes. Then, a Reed-Solomon code is applied on the authentication information, which consists of a polynomial interpolation on $\rho n + 1$ positions and a polynomial evaluation in $n - \rho n - 1$ positions. These tasks require $O(n \log n)$ field operations or $O(n \log n\ N \log^{O(1)} N)$ time, since both polynomial evaluation and interpolation for polynomials of degree at most n can be solved using $O(n \log n)$ field operations (thus, Reed-Solomon encoding requires a quasi-linear $O(n \log n)$ number of field operations). Observe that the use of a systematic Reed-Solomon code adds no extra computational cost.

Decoder: From Theorem 2.1, we have that $O(\beta^2 n^2 N) = \tilde{O}(n^2)$ field operations are required and thus $O(\beta^2 n^2 N^2 \log^{O(1)} N) = \tilde{O}(n^2)$ time is needed for the decoder to run. Also, for each of $O(1)$ candidate polynomials, we perform a polynomial evaluation at $\rho n + 1$ positions, thus $O(n \log n)$ field operations and so $O(n \log n N \log^{O(1)} N)$ time, and one signature verification. In total, we have $O(n^2 N^2 \log^{O(1)} N) = \tilde{O}(n^2)$ processing time and $O(1)$ signature verifications. Finally, $O(n)$ hash values are computed.

Communication Cost. The size of the authentication information is $\frac{n}{\rho n+1}(nh + s)$. That is, we have *constant communication overhead* per packet $\frac{nh+s}{\rho n+1} < \frac{h}{\rho} + \frac{s}{\rho n} = \frac{h}{\rho} + o(1)$. We see that $1/\rho$ hash values are included in each packet, with $\rho = \frac{\alpha^2}{(1+\epsilon)\beta} < 1$. The larger the value of ρ the smaller the authentication overhead.

Delay. As *delay*, we count the number of packets that the authenticator or decoder algorithm has to buffer. Of course, by definition, any authentication scheme according to our model needs to process n packets. However, delay is a cost parameter that is useful even in our model, since it captures the ability of the authenticator or the decoder to process packets in an *on line* fashion. In our scheme the sender processes n packets and the receiver processes βn packets in the worst case. However, the receiver can invoke an decoding procedure only after $\rho n + 1$ or αn packets have been received.

In particular, the receiver can try to compute the authentication information exactly after $\rho n + 1$ packets are received: the used code is systematic and the first $\rho n + 1$ symbols of $E(S)$ equal S (where S is the authentication information). Of course, we need no packet loss to occur among these packets. If the correct polynomial is computed (and verified) from the first $\rho n + 1$ packets, the authentication information is computed without any decoding overhead. Similarly, the receiver can try to compute the authentication information after αn packets are received: this time the decoder runs completely, but computation is a less expensive, and if the correct authentication information is computed, no attack is in process and the delay is αn. Otherwise, if no polynomial can be verified, the receiver is under attack and βn delay is required in the worst case. In other words, our scheme can distinguish between the less expensive *detection* of an attack by an adversary from the more expensive *verification* of the valid received packets. We believe that this feature is desirable, for less computational effort is spent when no adversary acts.

5.1. Tuning and Extensions

Given specific values of the survival rate α and flood rate β of the network, the parameter ρ, which controls the communication overhead, can be tuned by the tolerance parameter ϵ. This gives one degree of freedom in implementing the exact encoding-decoding procedures. Namely, bandwidth consumption can be decreased at the cost of increasing by a constant factor the time complexity and vise versa. A realistic deployment of our scheme can consider α and β as an additional degree of freedom: early packet streams (groups of packets) are encoded for bigger values of α and smaller values of β. Depending on the observed network's behavior, the network parameters can be later adjusted to a new desired level of security. Table 1 shows the communication overhead per packet for specific values of α, β and ϵ.

Independently of the choice of parameters, our scheme can be further modified in two ways, achieving different trade-offs between communication cost and computational efficiency. First, we can decrease the communication overhead, by applying the technique of [20]. The idea is that, since (at least) αn packets are guaranteed to be received intact, a significant portion of the authentication information is obtained by the decoder for free and without decoding: the (at least) αn hash values of the valid packets. Thus, less authentication information can be used and less redundancy is added to packets. To implement this idea, one has to encode the n hash values appropriately and, thus, Reed-Solomon codes are applied twice. Interestingly, as opposed to the case of erasure codes [20], in our case where Reed-Solomon error correcting codes are used, the decrease of the communication overhead occurs only for appropriate ranges of values for the network parameters α and β.

In particular, let $\{X, X'\} \leftarrow C[n, k+1]_q(X)$ denote the application of systematic Reed-Solomon code $[n, k+1]_q$ on word X, where X' is the added redundancy. Also let $H = h_1 \circ \ldots \circ h_n$ be the hash values of the n packets. We get the modified scheme by encoding $\{H, H'\} \leftarrow C[\gamma n, n+1]_{q_1}(H)$ and then $\{A, A'\} \leftarrow C[n, \rho n+1]_{q_2}(A)$, where $A = H' \circ \mathsf{Sign}_{SK}(H)$, $\gamma = 1 - \alpha + \sqrt{(1+\epsilon)\beta}$, $q_1 = 2^h$, $\rho = \frac{\alpha^2}{(1+\epsilon)\beta}$ and $q_2 = 2^{\lceil \frac{|A|}{\rho n+1} \rceil}$. As in our basic scheme, $A \circ A'$ is split in n equal shares A_i and packet p_i corresponds to authenticated packet $a_i = \mathit{GID} \circ i \circ p_i \circ A_i$. At the decoder, by the network reliability (at least αn packets will be valid) it is guaranteed that a constant size list of candidate strings for $H' \circ \mathsf{Sign}_{SK}(H)$ is produced; also, list-decoding is transformed to unambiguous decoding by verifying a constant number of signatures. Furthermore, the receiver is always capable to list-decode the packet hashes H. If in total δn packets reach the receiver, $\delta \leq \beta$, then Corollary 2.3 holds, since, $t \geq \alpha n + (\gamma - 1)n \geq \sqrt{(1+\epsilon)\delta}n$. The per packet communication overhead of this scheme is $\frac{(\gamma-1)h}{\rho}$. When $\gamma < 2$, with this scheme we save in communication overhead. That is, for network parameters α and β in appropriate ranges so that $\beta < \frac{(\alpha+1)^2}{1+\epsilon}$ we can decrease the communication cost by the constant factor γ at the cost of increasing the computational cost by roughly a factor of 2, since two applications of Reed-Solomon codes are required.

Also, by decreasing the field size, we can reduce the cost of performing field operations. For instance, we could split the authentication information into $\gamma\rho n + 1$ substrings of size ℓ, $\gamma > 1$ (e.g., $\gamma = 10$), consider each substring as a field element in $\mathbb{F}_q$, with $q = 2^\ell$, encode with a $[\gamma n, \gamma\rho n+1]_q$ Reed-Solomon code, and split the augmented authentication information into n pieces (each of γ field elements). In this way, the communication cost stays the same, but field operations become faster. The number of field operations at the encoder or decoder is increased by only a constant factor. Depending on the hardware architecture, this modification may be useful. A drawback here is that one injected packet by the adversary is now affecting the decoding algorithm by a factor γ.

5.2. Comparison with other schemes

We compare our schemes against various classes of proposed multicast authentication schemes.

Sign-All and Merkle Tree Schemes. The sign-all and Merkle-tree [28] authentication schemes are resilient to fully adversarial networks. The sign-all scheme involves one signature (resp. verification) operation per packet and a communication overhead that is equal to the signature size. Depending on the specific signature scheme in use, the parameters of our scheme or the architecture, both communication and computational costs of our scheme are comparable to the corresponding costs of the sign-all scheme.

Very short signature schemes have recently been proposed [3]. While the length of a signature can be as low as 160 bits, the security of this signature scheme is only proven in the random oracle model, and only under a strong assumption (Diffie-Hellman assumption in gap-DH groups, see Boneh and Franklin [2] for more on these groups). Signing every packet with this short signature, therefore, has a communication advantage over our construction, but loses in provable security. On the other hand, signing every packet with a provably secure signature, such as the the Cramer-Shoup [6] signature or its modification due to Fischlin [9], will add about 500 bytes to each packet — which is more than what we have for reasonable α and β.

α	β	ϵ	$1/\rho$	cost c (bytes)	α	β	ϵ	$1/\rho$	cost c (bytes)
0.33	1.5	0.1	15.15	303	0.5	1	0.01	4.04	81
0.5	1.5	0.1	6.6	132	0.5	2	0.01	8.08	162
0.75	1.5	0.1	2.93	59	0.5	3	0.01	12.12	243
0.33	1.5	0.5	20.66	414	0.5	1	0.1	4.4	88
0.5	1.5	0.5	9	180	0.5	2	0.1	8.8	176
0.75	1.5	0.5	4	80	0.5	3	0.1	13.2	264

Table 1. Communication cost c per packet for various values of the survival rate α, flood rate β and tolerance parameter ϵ. We assume the use of the SHA-1 hashing algorithm, that is, $h = 20$ bytes. The communication cost should be compared with the size s of the signature in use (e.g., an RSA signature with $s = 256$ bytes). Recall that $\rho = \frac{\alpha^2}{(1+\epsilon)\beta}$ is the rate of the code in use and that $c = \frac{h}{\rho} = \frac{h\beta(1+\epsilon)}{\alpha^2}$.

Additionally, signing every packet is undesirable in practice. Indeed, by signing every packet separately we lose both in efficiency and in architecture design since the secret key operations are computationally expensive and require extra need of security. Invoking a signature operation involves fetching the private key and temporarily storing it in the main memory of the system. When secret key operations are performed at high rates, the secret key resides almost exclusively in the memory of the system increasing the danger of the key being compromised to other running processes in the system. Special-purpose hardware can be used to overcome this problem, but of course at a higher cost. In terms of secure architecture design costs, and also for provable security or efficiency reasons, the sign-all approach is inferior to ours.

Finally, since one signature verification must be performed for each received packet, valid or not, the sign-all solution suffers by the following denial-of-service attack at the receiver: by injecting invalid packets an adversary can increase the computation resources spent at the receiver for signature verifications. In our scheme, where signature dispersal is used, no such attack is possible.

On the other hand, the Merkle-tree scheme has better time complexity than our scheme. For a group of packets of size n, only $2n$ hash computations and one signature computation (resp. verification) are performed at the sender (resp. receiver[5]). However, the Merkle-tree scheme has communication cost that grows with the number of packets, thus, this scheme is not scalable. Our scheme is efficient in terms of communication cost: packets have constant authentication overhead.

5 Note that at the receiver, by appropriately caching hash values, only one signature verifi cation is needed: once the fi rst valid packet is verifi ed, its (authenticated) hashes are stored and subsequent packets need only be verifi ed with respect to the hashes they carry. Because of that, injected packets do not necessarily cause signature verifi cations.

Graph-Based Schemes. These schemes [12, 22, 19, 27] assume the reliable receipt of a signature packet. However, a fully adversarial network will capture the signature packet and invalidate the scheme. Even if the signature packet is assumed to arrive intact, any efficient scheme in terms of communication overhead (i.e., with constant overhead for packet) will have the undesirable property that $O(1)$ critical packets can be adversarially chosen to disconnect from the authentication chain the signature node (packet). In the piggybacking scheme in [19], this number of critical packets can be $O(n)$ at the expense of a communication overhead of $O(n)$ per packet. Our scheme does not have these drawbacks since the signature is dispersed among all the packets. As opposed to graph-based authentication where the authentication of a packet crucially depends on other packets (with packets closer to the signature packet being more important), our scheme is symmetric in this context: all packets share the authentication information.

Erasure-Code Schemes. These schemes [21, 20] make use of erasure codes to tolerate packet losses, up to a constant fraction. However, no packet injections are tolerated: a single injected packet suffices to fail the decoding procedure. For networks where packets get only lost, they perform slightly better than our scheme in terms of communication cost and time complexity. This is due to the fact that erasure codes are more efficient than error-correcting codes in terms of time complexity and space requirement. Moreover, erasure codes can tolerate more erasures than the theoretical limit $d/2$ for error correcting codes (d is the diameter of the code). In [15], the authors address the vulnerability to packet injections that any scheme based on erasure-codes has, but their proposed scheme has high communication overhead and is not scalable, because a Merkle hash tree is used to “filter out” the injected packets (and thus the communication cost is $O(\log n)$).

Other Schemes. TESLA [22] and the scheme by Xu and Sandhu [29] have very different assumptions from our model. They are both based on MACs and on strong time-synchronization requirements about the nodes of the networks that do not fit our model. For instance, in [29], the routers of the networks are considered trusted entities.

Table 2 summarizes the above discussion, were selected schemes are compared with our scheme. In particular, we consider two graph-based authentication schemes, one of constant degree (expander construction [27]) and one of $O(n)$ degree (piggybacking scheme with parameterized performance [19], where we assume a constant number of classes), and one erasure scheme (optimized in terms of communication scheme [20]).

6. Conclusion

In this paper, we propose a new general framework for the multicast authentication problem, where the network is controlled by an adversary that has great, yet not unlimited, power in modifying the transmitted stream. Our model is realistic in terms the of adversarial model and the security assumptions. The limitations on the adversary's power, characterized by the survival and flood rates, exclude from consideration only degenerate cases, where the authentication problem actually disappears.

Our work establishes a new direction in multicast authentication by going beyond erroneous networks and addressing fully adversarial networks. Our authentication scheme is efficient, lightweight and practical. It is as secure as the "sign-all" solution, but more efficient in both computational effort and communication overhead. Its constant communication overhead makes it scalable and preferable to the approach by Wong and Lam [28]. When compared with this Merkle-tree based scheme, the $O(n^2)$ time complexity of our scheme is a shortcoming. However, it is possible that in practice this may not be a serious concern. Additionally, our scheme can be tuned by the network parameters α and β and distinguishes between the less expensive detection of an attack by the adversary and the more expensive task of verification.

Open problems to address in future work are as follows. First, we would like to investigate the practical performance of our multicast authentication approach by implementing it and conducting an experimental study. Also, a natural question to explore is whether other classes of error correcting codes can be employed in our framework.

Moreover, in this paper we showed a connection between coding theory and cryptography. In particular, we employed cryptographic primitives to unambiguously list-decode an error correcting code. It would be interesting to study whether there are other connections between the two areas. Finally, we would like to explore the use of our technique in other authentication problems.

Acknowledgments

We would like to thank Philip Klein for useful discussions and also Christoph Schuba and the anonymous referees for their detailed comments.

This work was supported in part by the National Science Foundation under grants CCR–0311510 and IIS–0324846 (Information Technology Research program).

References

[1] D. Boneh, G. Durfee, and M. Franklin. Lower bounds for multicast message authentication. In B. Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 437–452. Springer Verlag, 2001.

[2] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Verlag, 2001.

[3] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology, ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer-Verlag, 2001.

[4] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *IEEE INFOCOM*, pages 708–716, 1999.

[5] A. C.-F. Chan. A graph-theoretical analysis of multicast authentication. In *Proc. 23rd International Conference on Distributed Computing Systems – ICDCS*, pages 155–162, 2003.

[6] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In *Proc. 6th ACM Conference on Computer and Communications Security*, pages 46–52. ACM press, Nov. 1999.

[7] T. Cucinotta, G. Cecchetti, and G. Ferraro. Adopting redundancy techniques for multicast stream authentication. In *Proc. 9th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003)*, pages 189–201, 2003.

[8] Y. Desmedt, Y. Frankel, and M. Yung. Multi-receiver/multi-sender network security: Efficient authenticated multicast/feedback. In *IEEE Conference on Computer Communications — INFOCOM '92*, pages 2045–2054. IEEE-Press, 1992.

[9] M. Fischlin. The Cramer-Shoup strong-RSA signature scheme revisited. In Y. Desmedt, editor, *Public Key Cryptography - PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[10] R. Gennaro and P. Rohatgi. How to sign digital streams. In B. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 180–197. Springer Verlag, 1997.

	Sign-all	Merkle [28]	GB [27]	GB [19]	Erasure [20]	Our
Delay S	1	n	n	n	n	n
Computation S						
Sign	n	1	1	1	1	1
hash	—	$2n$	$O(n)$	$O(n^2)$	n	n
field op	—	—	—	—	$O(n \log n)$	$O(n \log n)$
Communication	ns	$n(s + h \log n)$	$O(nh)$	$O(n^2 h)$	αhn	$\frac{h\beta(1+\epsilon)}{\alpha^2} n$
Delay R	1	1	n	n	n	βn
Computation R						
Verify	n	1	1	1	1	$O(1)$
hash	—	$2n$	$O(n)$	$O(n^2)$	n	βn
field op	—	—	—	—	$O(n^2)$	$\tilde{O}(n^2)$
Secret key protection	—	●	●	●	●	●
Resiliency						
Chosen packet loss	●	●	—	●	●	●
Chosen packet injection	●	●	●	●	—	●
Signature dispersal	●	●	—	—	●	●

Table 2. Comparison of selected multicast authentication approaches with respect to various aspects of efficiency, security and resiliency. By S (resp. R) we denote the sender (resp. receiver), Sign **denotes a signature operation,** Verify **denotes a signature verification,** hash **denotes the total hashing cost, where we consider that the complexity of hashing a string is a linear function of the string size. Also, we use the following notation: n is the number of packets in the data stream, s is the signature size and h is the hash size. Both the communication overhead and the computational costs refer to n packets.**

[11] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, Apr. 1988.

[12] P. Golle and N. Modadugu. Authenticating streamed data in the presence of random packet loss. In *Network and Distributed System Security Symposium —NDSS '01*, pages 13–22, 2001.

[13] V. Guruswami. *List Decoding of Error-correcting Codes*. PhD thesis, Massachusetts Institute of Technology, Boston, MA, 2001.

[14] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. In *IEEE Transactions on Information Theory*, pages 45:1757–1767, 1999.

[15] C. Karlof, Y. Li, and N. Sastry. Authenticated block streams using error detecting erasure codes. Manuscript. Available from `http://www.cs.berkeley.edu/~nks/edec/bcast-class.pdf`, 2003.

[16] M. Luby. LT codes. In *43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS '02)*, 2002.

[17] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann. Practical loss-resilient codes. In *29th Annual ACM Symposium on Theory of Computing (STOC '97)*, pages 150–159, 1997.

[18] S. Micali. 6.875: Introduction to cryptography. MIT course taught in Fall 1997.

[19] S. Miner and J. Staddon. Graph-based authentication of digital streams. In *IEEE Symposium on Security and Privacy*, pages 232–246, 2001.

[20] A. Pannetrat and R. Molva. Effi cient multicast packet authentication. In *Proc. Network and Distributed System Security Symposium — NDSS '03*, 2003.

[21] J. M. Park, E. K. P. Chong, and H. J. Siegel. Effi cient multicast packet authentication using signature amortization. In *IEEE Symposium on Security and Privacy*, pages 227–240, 2002.

[22] A. Perrig, R. Canetti, J. Tygar, and D. Song. Effi cient authentication and signing of multicast stream over lossy channels. In *IEEE Symposium on Security and Privacy*, pages 56–73, 2000.

[23] M. O. Rabin. Effi cient dispersal of information for security, load balancing, and fault tolerance. *Journal of ACM*, 36(2):335–348, 1989.

[24] I. S. Reed and G. Solomon. Polynomial codes over certain fi nite fi elds. *SIAM Journal of Applied Mathematics*, 8(2):300–304, 1960.

[25] P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *Proc. 6th ACM Conference on Computer and Communications Security*, pages 93–100. ACM, 1999.

[26] G. J. Simmons. Authentication theory/coding theory. In *Proceedings of the Conference on Advances in Cryptol-*

ogy (CRYPTO'84, Santa Barbara,CA), LNCS 196, Springer-Verlag, pages 411–431, 1984.

[27] D. Song, D. Zuckerman, and J. D. Tygar. Expander graphs for digital stream authentication and robust overlay networks. In *IEEE Symposium on Security and Privacy*, pages 258–27, 2002.

[28] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. In *Proceedings of the 1998 International Conference on Network Protocols (ICNP '98)*, pages 198–209, Austin, Texas, Oct. 1998.

[29] S. Xu and R. Sandhu. Authenticated multicast immune to denial-of-service attack. In *Proc. ACM Symposium on Applied Computing*, pages 196–200, Madrid, Spain, Mar. 2002.