Authenticated Error-Correcting Codes with Applications to Multicast Authentication

ANNA LYSYANSKAYA, ROBERTO TAMASSIA, and NIKOS TRIANDOPOULOS
Brown University

We consider the problem of authenticating a stream of packets transmitted over a network controlled by an adversary who may perform arbitrary attacks on the stream: He may drop or modify chosen packets, rearrange the order of the packets in any way, and inject new, random, or specially crafted packets into the stream. In contrast, prior work on the multicast authentication problem has focused on a less powerful adversarial network model or has examined a considerably more restrictive setting with specific timing or structural assumptions about the network.

We model the ability of the network to modify a stream of n packets with two parameters: the *survival rate* α ($0 < \alpha \leq 1$) denoting the fraction of the packets that are guaranteed to reach any particular receiver unmodified and the *flood rate* β ($\beta \geq 1$) indicating the factor by which the size of the received stream at any particular receiver may exceed the size of the transmitted stream. Combining error-correcting codes with standard cryptographic primitives, our approach 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 to each transmitted packet only a small amount of authentication information, proportional to β/α^2. We prove the security and correctness of our scheme and analyze its performance in terms of communication overhead and computational effort at the sender and the receiver. Our results demonstrate how list decoding can be transformed into unambiguous decoding in the public-key model and the bounded computational model for the underlying communication channel. Overall, our technique provides an authenticated error-correcting code of independent interest that may be useful in other settings.

Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms: Security, Reliability, Verification, Design

Additional Key Words and Phrases: Authentication, data stream, digital signature schemes, error-correcting codes, information integrity, list decoding, multicast security

Work first presented at the IEEE Symposium on Security & Privacy as [Lysyanskaya et al. 2004]. This work was supported in part by the National Science Foundation under grants CCR–0311510, IIS–0324846, IIS–0713403, and OCI–0724806.

Authors' address: Brown University, Dept. of Computer Science, Box 1910, Providence, RI 02912.

DOI 10.1145/1698750.1698757 http://doi.acm.org/10.1145/1698750.1698757

ACM Reference Format:
Lysyanskaya, A., Tamassia, R., Triandopoulos, N. 2010. Authenticated error-correcting codes with applications to multicast authentication. ACM Trans. Info. Syst. Sec. 13, 2, Article 17 (February 2010), 34 pages.
DOI = 10.1145/1698750.1698757 http://doi.acm.org/10.1145/1698750.1698757

1. INTRODUCTION

Multicast transmissions of data streams are widely being used in applications such as peer-to-peer networks, music and video distribution, stock quotes, surveillance, and environmental monitoring. We address the problem of verifying the integrity of the received stream in the presence of transmissions errors and malicious attacks.

Multicast authentication is especially challenging, since the transmission does not have to be reliable and packet losses must be tolerated (e.g., IP multicast is implemented with a best-effort delivery mechanism over the UDP transport protocol). Thus, the authentication should verify as many as possible of the received packets without assuming the availability of the entire original stream. Besides commonly occurring transmission errors, we also want to defend against an adversary who can modify the stream (e.g., controlling a rogue router or spoofing packets).

Therefore, 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. The authentication mechanism should be as general as possible and should not depend on any specific assumptions about the underlying network. Although this problem has been extensively studied, no formal definition has been given for this general version. 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., Perrig et al. [2000], Golle and Modadugu [2001], Miner and Staddon [2001], and Song et al. [2002]) or no packet injections occur (e.g., Park et al. [2003] and Pannetrat and Molva [2003]), or relatively strong and hard to meet conditions are assumed about the network behavior (e.g., Perrig et al. [2000, 2001]). Thus, most of the previously proposed schemes rely on less-general network models, tolerate only 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 article, 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. Using this network model, we formally define the authentication problem for multicast transmissions and describe the notions of correctness and security that any authentication scheme should respect. 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. The use of error-correcting codes for multicast authentication was inspired by the previous use of erasure codes, as in Park et al. [2003], and Pannetrat and Molva [2003], for this problem. We essentially design authenticated error-correcting codes that constitute a new authentication tool, useful in many settings. Moreover, enhanced with cryptographic primitives, our error-correction encoding derives an interesting connection between cryptography and coding theory. Namely, a corollary of our results shows that in the setting where the underlying communication channel is modeled as a computationally bounded (i.e., polynomial-time) adversary, list decoding can be directly transformed to unambiguous decoding by appropriately employing simple cryptographic primitives in the encoding and decoding design (see Corollary 4.4.3). This is perhaps the first result in the literature connecting coding theory and cryptography.

In the rest of this section, we introduce our model, summarize our contributions, and present previous work on multicast authentication. The organization of the rest of the article is as follows. The cryptographic primitives and error-correcting codes used in our work are reviewed in Section 2. In Section 3, we describe in detail our adversarial network model and multicast authentication framework. Section 4 describes the construction of our authenticated error-correcting code and 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. In practice, the survival and flood rates are functions of the network structure rather than the amount of transmitted data and thus, are expected to be constants.

As we will see in Section 1.2, similar assumptions on the survival of correct packets and the amount of injected packets are typical in the multicast authentication literature. Note that 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. In our approach, if the (α, β) assumption does not hold, the attacker prevents the receiver from validating the stream but cannot convince the receiver to accept a corrupted packet—in this case, security is preserved but denial of service occurs.

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 (in particular, our scheme is based on efficient list-decoding techniques due to Guruswami and Sudan [1999], as we will see in detail in Section 2). In essence, we design an authenticated Reed-Solomon error-correcting code that constitutes a new, powerful, and general-purpose authentication tool. This last feature of our scheme provides a new interesting connection between coding theory and security. In particular, we show how, in the public-key model and the bounded computational model for communication channels, list decoding can be transformed into unambiguous decoding.

—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 approaches that provide security in our adversarial model is (i) the inefficient "sign-every-packet" solution, which involved either singing each packet individually or using a Merkle hash tree as in the scheme by Wong and Lam [1999], and (ii) a recently proposed scheme by Karlof et al. [2004] that uses signature dispersal and a Merkle hash tree. 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. Our scheme, by amortizing one signature per a stream of size n, suffers from no such problem. On the other hand, the tree-based authentication schemes [Karlof et al. 2004; Wong and Lam 1999] have the drawback that the per-packet communication overhead grows logarithmically with the number of packets sent. Indeed, each packet of a stream of size n carries authentication information of size $O(\log n)$. In contrast, our scheme achieves per-packet communication overhead independent of n and, thus, more efficiency and scalability.

1.2 Prior and Related Work

Previous work on multicast authentication considers both unconditionally secure schemes (e.g., Simmons [1984], and Desmedt et al. [1992]), which tend to be less practical, and computationally secure schemes, which we overview next by appropriately categorizing them according to the underlying authentication technique in use.

MAC-Based Approaches. Various approaches use message authentication codes (MACs) and secret-key cryptography. A naïve solution here is to have all the receivers sharing a secret key and include a MAC into every packet sent, but this scheme is not secure, as any user can spoof packets, or alternatively to have each receiver sharing her own secret key with the sender and add to each packet a MAC for every receiver, but this scheme is not scalable as the communication cost is high.

Canetti et al. [1999] described a MAC-based scheme that is secure with high probability against any coalition of w corrupted users, where $O(w)$ MACs are appended to each packet, this scheme is not fully scalable due to its communication overhead. Perrig et al. [2000, 2001] presented another MAC-based scheme, TESLA, 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. Note also that although each packet carries a separate authentication tag, if too few packets survive the packet losses, the scheme is not functional as the key chaining is destroyed as well. Two MAC-based schemes that make explicit use of the topology of a multicast tree and assume trusted routers were proposed by Xu and Sandhu [2002]. 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. [2001] generalized MACs to a multicast setting by defining a new primitive for multicast authentication called multicast MAC (MMAC). Satisfying certain correctness and security constraints, a MMAC is a triplet of algorithms $(key\_gen, mac\_gen, mac\_ver)$ where: key_gen produces secret keys sk for the sender and $rs_1, \ldots, rs_n$ for the receivers; $mac\_gen(M, sk)$ computes an authentication tag τ for message M and $mac\_ver(M, \tau, rk_i)$ checks whether τ is an authentication tag of message M. In their work, Boneh et al. studied the existence of efficient MMACs, that is, MMACs with short tag τ, and showed

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, which brings us to signature amortization (see later discussion). The result was also extended to the case where the adversary possesses a limited number of the receivers' keys and a lower bound on the length of the authentication tag was derived. The construction by Canetti et al. [1999] meets this bound, whereas TESLA by Perrig et al. [2001, 2000], using time synchronization and one digital signature for bootstrapping, is not an exact MMAC scheme.

In view of the previous result, research also focused on building faster signature schemes for signing every packet separately. Work on this direction includes: the use of one-time signatures by Gennaro and Rohatgi [1997] for multicast transmission in reliable communication channels, the use of k-time signatures by Rohatgi [1999] to speed-up the signing rate, and the BiBa broadcast protocol by Perrig [2001].

Signature Amortization. Many approaches use the technique of signature amortization, where a single digital signature is used for the authentication of multiple packets. The first scheme that uses this method over a hash chain was by Gennaro and Rohatgi [1997]: Each packet p_i is augmented with authentication information a_i, recursively defined as the hash of $p_{i+1} \circ a_{i+1}$ ($\circ$ denotes concatenation), and the augmented packet $p_1 \circ a_1$ is digitally signed. This scheme has constant authentication overhead per packet but does not tolerate packet losses. A Merkle hash tree was used by Wong and Lam [1999] 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 arbitrarily high packet loss rates, but as any scheme with packets carrying separate authentication information, it implicitly assumes an upper bound on the injected packets; otherwise, a simple denial-of-service attack can be performed as any injected invalid packet enforces a separate packet verification. Overall, this scheme requires one signature verification per each received packet and has logarithmic communication overhead per packet. In contrast, our approach, which also uses signature amortization, has constant per-packet communication overhead and requires only $O(1)$ signature verifications amortized over n packets.

Graph-Based Authentication. This method 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., not adversarially). 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.

Perrig et al. [2000] selected G to be an augmented-chain graph, consisting of a path plus additional edges that connect vertices at various distances. Golle and Modadugu [2001] proposed an augmented chain graph that specifically tolerates bursty packet losses. Random graphs and a scheme resilient to multiple bursty losses were studied by Miner and Staddon [2001] and expander graphs were used by Song et al. [2002]. By modeling packet losses in a probabilistic setting (e.g., with some fixed probability and independently of others, each packet gets lost), all graph-based schemes implicitly assume that a constant fraction of the transmitted packets reach any given receiver. This is in total agreement with our survival rate α.

Erasure Codes. Park et al. [2003] and Pannetrat and Molva [2003] used erasure codes (e.g., Rabin [1989] and Luby et al. [2001]) for multicast authentication to tolerate adversarially chosen packet losses and disperse one signature over a group of packets. In particular, information sufficient—if reconstructed at the receiver—to authenticate the received packets is encoded using an erasure code so that delivery of a constant fraction of packets guarantee the successful decoding of this information. Both constructions are efficient in terms of communication cost. The two schemes only differ in that in Pannetrat and Molva [2003], 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. Park et al. [2003] identified a special case of this packet-injection problem and suggested the use of distributed fingerprints [Krawczyk 1993] to tolerate a small number of symbol modifications of the erasure-encoded information. However, the proposed scheme lacks efficiency, since distributed fingerprints are used on top of the erasure encoding, and, more importantly, it collapses for a specific packet-injection attack, where more than one symbol claims to be a specific symbol. We note that our adversarial model includes such attacks: Many injected packets may pretend to be a specific original packet.

To mitigate the packet-injection problem that all erasure-code schemes have, Karlof et al. [2004] introduced the notion of *distillation codes*, which tolerate both erasures and pollution of symbols. They realized such codes by employing a membership authentication scheme—namely, Merkle's hash tree—along with an erasure code (as in Park et al. [2003]). Symbols are erasure encoded as usual in groups of size n, but also appended by a group-inclusion witness. Using these verifiable witnesses, the received symbols are partitioned into the appropriate pre-encoded groups, which are then decoded and examined to be authentic. This approach tolerates packet injections and erasures with not constant per-packet

communication overhead (each packet carries a witness of size $O(\log n)$), and suffers from the cumbersome decoding procedure at the receiver: The partition operation adds a considerable amount of hashing and erasure decodings. Gunter et al. [2004] proposed a scheme that is based on erasure codes and tolerates packet injections that occur up to a certain transmission rate. The scheme involves the use of three different streams (data stream, hash and parity stream, signature stream), where signatures are verified selectively and injected packets are filtered out by exhaustively processing all received packets. This scheme offers only probabilistic authentication guarantees and tolerates no adversarial packet losses. We note that by construction, all schemes based on erasure codes implicitly assume a constant packet survival rate; otherwise, reconstruction of the authentication information is not guaranteed.

Related Work. Krohn et al. [2004] have used erasure encoding techniques in combination with homomorphic hashing for the on-the-fly verification of erasure-encoded blocks. However, their scheme relies on the following strong assumption: The receiver must know in advance the cryptographic hashes of the transmitted shares.

Techniques similar to ours have been used by Micali et al. [2005] in subsequent work to study error correction in the computational model. Their main result confirms the main finding of ours, namely that in the cryptographic setting unambiguous decoding can be achieved beyond the classical information-theoretic bounds. Also, results for codes with binary alphabets and optimality results are presented.

Interestingly, the authentication capabilities of our crypto-enhanced error correction techniques have been recently applied by Goldberg [2007] to improve the robustness of protocols for private information retrieval. In his work, Goldberg also presents an efficient implementation of the authenticated version of the list decoder by Guruswami and Sudan [1999]. We note that since their initial development, these list-decoding algorithms have been significantly improved (e.g., McEliece [2003]).

2. PRELIMINARIES

Before defining the cryptographic and coding primitives that we use in our construction, we first introduce some notation. Let A be an algorithm. By $A(\cdot)$, we denote that A has one input and by $A(\cdot, \ldots, \cdot)$ 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. That is, 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, where we additionally assume that the query tape Q is always the first output of any such algorithm $A^O(\cdot)$. 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; also, PPT stands from probabilistic polynomial time.

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. By $\Pr[\{x_i \leftarrow A_i(y_i)\}_{1\le i\le n} : b(x_n)] = \alpha$, we denote 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.

2.1 Cryptographic Primitives

The following definition is due to Goldwasser et al. [1988] 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 2.1.1 (*Signature Scheme*). The set of PPT algorithms $(G(\cdot), \mathsf{Sign}_{(\cdot)}(\cdot), \mathsf{Verify}_{(\cdot)}(\cdot,\cdot))$, where G is the key generation algorithm, Sign is the signing algorithm, and Verify the verification algorithm, constitutes a 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}. That is, for all values m and k

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

Security. Even if an adversary has oracle access to the signing algorithm that provides signatures on arbitrarily chosen messages, the adversary cannot create a valid signature on a message not explicitly queried. That is, for all families of PPT oracle Turing machines $\{A_k^{(\cdot)}\}$, there exists a negligible function[1] $\nu(k)$ such that

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

For completeness, we give the standard definition (see, e.g., Goldreich [2004]) of a family of collision-resistant hash functions.

Definition 2.1.2 (*Collision-resistant Hash Function*). Let $\mathcal{H}$ be a PPT algorithm that, on input 1^k, outputs an algorithm $H \ : \ \{0,1\}^* \mapsto \{0,1\}^k$. Then $\mathcal{H}$ defines a family of collision-resistant hash functions if the following two hold.

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

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

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

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

2.2 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 $[q] = \{1, 2, \ldots, q\}$. An error-correcting code $[n, \ell]_q$, $\ell \leq n$, is a function[2] $C : [q]^\ell \rightarrow [q]^n$ that takes as input an ℓ-length message, or *word*, x over $[q]$ and outputs a longer n-length *codeword* $C(x)$ over the same alphabet. That is, an error-correcting code processes ℓ characters (symbols) in $[q]$ and adds redundancy to form n characters. If only redundancy is added by C such that the ℓ first symbols of $C(x)$ form word x, then the code is called *systematic*.

This processing and added redundancy help correcting up to e errors of the codeword $C(x)$: given a received word $y \in [q]^n$ such that y and $C(x)$ differ in at most e characters, one can unambiguously correct y to $C(x)$ and decode to x. The value e depends on the exact choice of code C. For unambiguous decoding, e is always bounded from above by $d/2$, that is, $e < d/2$, where d is the *distance* of the code, defined as follows. The distance of two codewords of C is the number of positions where their symbols differ and the (minimum) distance of code C is the minimum distance of two codewords of C, measured over all pairs of codewords.

List decoding allows to correct a number of errors that is beyond the half-the-distance bound discussed previously in the text. Handling even more than $d/2$ errors comes at a price, though. In fact, one can ambiguously correct beyond this bound. That is, given that $C(x)$ is received as word $y \in [q]^n$ and $C(x)$ and y differ in at most e positions with $e \geq d/2$, list decoding provides a list of candidate initial words in $[q]^\ell$, such that x belongs in this list. In general, as e grows, the size of the output list grows as well and, typically, we put an upper bound on e so that list decoding is efficient (i.e., is performed in polynomial time) and the list size is reasonable to work with (i.e., is constant).

Reed-Solomon codes [Reed and Solomon 1960] are a family of codes that are based on properties of univariate polynomials over finite fields. An $[n, k+1]_q$ Reed-Solomon code, $k < n \leq q$,[3] has distance $d = n - k$ and, as long as the number of errors e is, at most, $(d-1)/2 = (n-k-1)/2$, one can unambiguously decode in quadratic time in n [Welch and Berlekamp 1986] (in both field and bit operations). If $e > (n-k-1)/2$, then list decoding is considered feasible [Goldreich et al. 2000] (i.e., it can be performed in polynomial time) as long as $e \leq n - \sqrt{nk}$. We next define the $[n, k+1]_q$ Reed-Solomon code, where n, k, and

[2]There are more than one possible definitions. Here we model error-correcting codes as functions.
[3]Adopting the notation from Guruswami [2001], we use $[n, k+1]_q$ rather than $[n, k]_q$ to explicitly relate the code with the maximum degree k of the polynomials used in the encoding and decoding procedures (the importance of k will become clear later, when list decoding will be related to the polynomial reconstruction problem). We require that $k < n$ rather than $k + 1 < n$, allowing thus the extreme case, where, for $k = n - 1$, the code adds no redundancy at all.

q are positive integers and parameters of the code, with $k < n \leq q$, and where list decoding is considered. Here, we present 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 article 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 2.2.1 (*Reed-Solomon Code*). An $[n, k+1]_q$ Reed-Solomon code includes the following.

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

Encoder. The encoder is a function $C : \mathbb{F}_q^{k+1} \rightarrow \mathbb{F}_q^n$, with $n > k$, that specifically:

(1) uses a fixed canonical set $G_q = \{g_1, g_2, \ldots, g_{n-1}, g_n\}$ of n distinct field elements in $\mathbb{F}_q$, $G_q \subset \mathbb{F}_q$ (with a fixed ordering over elements g_i, $1 \leq i \leq n$, according to their index i);
(2) takes as input the parameters n and k, and $k+1$ points (g_i, y_i), $g_i \in G_q$, $1 \leq i \leq k+1$, and $y_i \in \mathbb{F}_q$ (this set of points is viewed as the *input word*);
(3) 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 $p(g_i) = y_i$, $1 \leq i \leq k+1$; and
(4) outputs points $(g_i, p(g_i))$, $1 \leq i \leq n$ (viewed as the input word's *codeword*).

We have that C is a systematic code. The ratio $\frac{k}{n} < 1$ is called the *rate* of the code.

(*List*) *Decoder*. The decoder takes as input parameters n and k, the maximum number of errors e that may occur, and n points (x_i, y_i), $x_i, y_i \in \mathbb{F}_q$, $1 \leq i \leq n$, and list decodes, that is, 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 no more than e values of i, $1 \leq i \leq n$; this list contains the polynomial computed in Step 3. Here, an error is defined as a change to the x-coordinate or y-coordinate of a point.

Note the direct correspondence between polynomials (produced by the decoder) and input words (processed by the encoder). The list of polynomials corresponds to a list of codewords, all being close enough to the distorted codeword (input of the decoder), which, in turn, maps to a list of candidate input words (each consisting of the set of points that result by evaluating a polynomial on the first $k+1$ elements of G_q). By definition, this list contains the correct input word of the encoder when the number of errors in the input points of the decoder does not exceed e.

For an $[n, k+1]_q$ Reed-Solomon code, we use a decoder that runs in polynomial time and is due to Guruswami and Sudan [1999]. The maximum number of errors that can be tolerated by this list-decoding procedure matches the theoretical bound of $n - \sqrt{kn}$. However, the decoding algorithm is prohibitively expensive (e.g., $O(n^{12})$) when the exact bound is met. By reducing this upper bound of the maximum number of errors to $n - \sqrt{(1+\epsilon)kn}$, $\epsilon \in (0, 1)$, a quadratic time decoding algorithm exists. We refer to this decoder as GS-Decoder. For this decoder, parameter ϵ controls how far away from the $n - \sqrt{kn}$ bound list

decoding can operate; thus, it also affects the performance of the decoder. We use the following result.

THEOREM 2.2.2 (GURUSWAMI AND SUDAN [1999]). *Consider a* $[n, k+1]_q$ *Reed-Solomon code. For any* $\epsilon \in (0,1)$*, given n points with at most* $n - \sqrt{(1+\epsilon)kn}$ *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.*

In practice, codes with constant expansion are used, where $k = \rho n$ for some constant $\rho < 1$, and ρ is the rate of the code. 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 2.2.2, we have the following, where by $\tilde{O}(\cdot)$ denotes that some logarithmic factors are omitted.

COROLLARY 2.2.3. *For any* $[n, \rho n+1]_q$ *Reed-Solomon code, for any constants* $\epsilon \in (0,1)$ *and* $\rho < 1$ *such that* $\sqrt{(1+\epsilon)\rho} \leq 1$*, on an input with at most* $(1 - \sqrt{(1+\epsilon)\rho})n$ *errors, GS-Decoder outputs a list of* $O(1)$ *size in* $\tilde{O}(n^2)$ *time, performing* $\tilde{O}(n^2)$ *field operations.*

PROOF. It follows directly from Theorem 2.2.2 for $k = \rho n$, $\rho < 1$ (using $\tilde{O}(\cdot)$ notation to hide constants and logarithmic on q factors). We need $\sqrt{(1+\epsilon)\rho} \leq 1$ so as the number of errors is not negative. □

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, $1 \leq i \leq n$. Parameter t is usually called *agreement*. Polynomial reconstruction and Reed-Solomon list decoding are equivalent problems [Guruswami 2001]. The following result is equivalent to Corollary 2.2.3.

COROLLARY 2.2.4. *For any constants* $\epsilon \in (0,1)$ *and* $\rho < 1$ *such that* $\sqrt{(1+\epsilon)\rho} \leq 1$*, 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.*

PROOF. It follows immediately from Corollary 2.2.3 by considering the equivalence between the two problems. The upper bound e of the number of errors guarantees agreement t of at least $n - e = \sqrt{(1+\epsilon)\rho}n$. We need $\sqrt{(1+\epsilon)\rho} \leq 1$ in order to be consistent with the requirement that $t \leq n$. □

What is crucial for the correctness and security of our scheme is that GS-Decoder [Guruswami and Sudan 1999; Guruswami 2001] solves the related polynomial reconstruction problem in the more general case where some of the input points may be vertically aligned, that is, where the x_i coordinates are not necessarily distinct. Viewing ϵ, as a parameter of the GS-Decoder, $0 < \epsilon < 1$, and considering the related polynomial reconstruction problem, we use $\mathsf{GSDecode}_\epsilon(n, k, t, \{(x_i, y_i) | 1 \leq i \leq n\})$ to denote that GS-Decoder runs with parameter ϵ to solve the polynomial reconstruction problem on degree k, agreement t, and n input points (x_i, y_i), $1 \leq i \leq n$.

3. NETWORK MODEL AND MULTICAST AUTHENTICATION FRAMEWORK

Considering data transmission in a multicast setting, a *sender*, the source of the data, transmits a data stream over an underlying "best-effort" network. Data packets are received by a large set of receivers. Without loss of generality, we focus our attention to one such receiver such that the two honest parties of the authentication protocol are the sender and the receiver. No guarantees about the delivery of the packets exist in general. Furthermore, the network is an adversary of great—yet not unlimited—power, acting in the bounded computational model. 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 (data) would ever get delivered, so our task would be hopeless. As mentioned in the introduction, we will restrict the adversary so that the transmitted stream is neither destroyed nor polluted by more than a constant fraction. In this regard, we view the cases where too few packets survive or too many packets are injected as two special denial-of-service attacks: Although reconstruction of the few valid packets is not achieved, yet no invalid packet is accepted as valid.

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. Each group of n packets is identified by a—unique per distinct group—group identification tag (GID). That is, packets of a group are marked with the corresponding GID . The sender should make sure not to use the same GID for another group of packets. Note that the existence of tag GID adds no new assumption about the transmitted stream. It corresponds to a means by which the packets can be grouped together. Even without an adversarial network, the sender should not reuse GIDs so as to make sure the recipient does not mix and confuse different groups of packets. In practice, a GID 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, for example, the size of a hash value (20 bytes for SHA-1).

3.1 The (α, β)-Network Model

As noted before, we model the network as an adversarial entity, that is, 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 maliciously chosen so that the adversary can cause the loss of any selected packets. Although the ability to tolerate packet losses has been widely considered an important property of multicast authentication schemes, only a few previous schemes (e.g., Wong and Lam [1999], Park et al. [2003], and Karlof et al. [2004]) tolerate adversarial losses, that is, the capability by the adversary to choose which packets are dropped and which survive. The adversarial loss model is the strongest and most realistic one, since it makes the least assumptions on how the traffic is routed. 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, except reactively in Karlof et al. [2004]. In contrast, we develop robust techniques for dealing with it. Finally, the adversary can arbitrarily modify, delay, or rearrange packets. Changing a packet corresponds to destroying it and injecting a new one.

An adversarial network modeled with the previously described capabilities is what we call a *fully adversarial network*. The following definition is standard.

Definition 3.1.1 (*Fully Adversarial Network*). A fully adversarial network is a network that is used for the transmission of a data stream and is controlled by a computationally bounded (i.e., PPT) adversary who can: (i) cause packets of her choice to be lost; (ii) inject packets (either random ones or with a specific malicious structure); and (iii) 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 modeling 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 article, both rates are considered to be constants. In a network with survival rate α, $0 < \alpha \leq 1$, if a stream of n packets is sent, at least αn packets of this stream will arrive at the receiver intact. In a network with flood rate β, $\beta \geq 1$, if a stream of n packets is sent, then the received stream will have at most βn packets.

Definition 3.1.2 (*Network Parameters*). Consider an adversarial network through which a sender transmits a stream of n packets. With respect to any particular receiver, the survival rate α, $0 < \alpha \leq 1$, is the minimum fraction of the packets that are guaranteed to reach the receiver unmodified, that is, at least αn packets in the received stream are valid, and the flood rate β, $\beta \geq 1$, indicates the maximum factor by which the size of the stream that reaches the receiver may exceed the size of the transmitted stream, that is, at most βn packets are in the received stream.

As shown in Section 1.2, the constant survival and flood rates are limitations on the adversary that have been explicitly or implicitly assumed in most of

the existing work on multicast authentication. We claim that the survival and flood rates of a network are reasonable parameters that better characterize the adversary's ability to modify the transmitted stream. In particular, the network's reliability, expressed by means of α and β, is not an assumption that affects the generality or the strength of our model; in contrast, it successfully emphasizes some intrinsic characteristics of the authentication problem we study. Indeed, the extreme cases where too few packets survive or too many packets are injected are both special, and perhaps less interesting, cases of the multicast authentication problem: At the receiver, no need for authentication really exists when $\alpha \to 0$ or $\beta \to \infty$. At the same time, our scheme presented in Section 4 is parameterized by the two rates α and β; that is, it operates correctly for any values of these two network parameters and securely for any adversary (even one not restricted to manipulating a $(\beta - \alpha)$ fraction of the packets).

A network with the previously mentioned 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. Although our discussion focuses on one particular receiver, for generality and completeness, we require that the network provides the same level of reliability (expressed by rates α and β) to any of the receivers.[4]

Definition 3.1.3 ((α, β)-*network*). An (α, β)-network is a fully adversarial network with survival rate α and flood rate β with respect to any receiver.

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 [Goldwasser et al. 1988]. This is not surprising since it is shown by Boneh et al. [2001] that the two problems are equivalent. A signature scheme consists of key generation, signature, and verification algorithms (see Definition 2.1.1). Similarly, we have key generation, authentication, and decoding algorithms, specified in the following text. Working in the public-key model, the key generation algorithm is run in advance to produce the private and public keys used by the involved parties, the sender and the receiver. The other 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.

Key generation. The key generation algorithm KeyGen is a PPT 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.

[4]This does not mean that the same data packets, that is, the same stream, reaches all the receivers.

Authenticator. The authenticator algorithm Auth takes as input the pair of secret and public keys (SK, PK), the group identification tag GID of the data stream, the number n of packets in the data stream, the survival rate α, the flood rate β and the data packets $DP = \{p_1, \ldots, p_n\}$, and it outputs the set $AP - \{a_1, \ldots, a_n\}$ of authenticated packets. We write: $AP \leftarrow \mathsf{Auth}(SK, PK, GID, n, \alpha, \beta, DP)$.

Decoder. The decoder algorithm Decode takes as input the public key PK, the group identification tag GID of the data stream, the number n of the original data packets, the survival rate α, the flood rate β, and the received packets $RP = \{r_1, \ldots, r_m\}$. The decoder is allowed to *reject* the input—for example, when more than $(\beta - \alpha)n$ packets are injected to the transmitted stream or when less than αn of the received packets are valid.[5] If it does not reject, it produces the output packets $OP = \{p'_1, \ldots, p'_n\}$. Some of these packets may be empty, each denoted by $\varnothing$, and correspond to the event that the decoder did not receive the corresponding authenticated packets. We write: $\{OP, \mathsf{reject}\} \leftarrow \mathsf{Decode}(PK, GID, n, \alpha, \beta, RP)$.

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 and in the correct order, that is, the corresponding data packets will be among the output packets, decoded with their correct indices. 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 nonauthenticated set of packets.

Definition 3.2.1 (*Multicast Authentication Scheme*). The set of PPT algorithms (KeyGen, Auth, Decode) constitutes an (α, β)-correct and secure multicast authentication scheme if no PPT adversary $\mathcal{A}$ can win nonnegligibly often in this game:

(1) A key pair is generated, that is, $(PK, SK) \leftarrow \mathsf{KeyGen}(1^k)$.
(2) The adversary $\mathcal{A}$ is given the public key PK as input and oracle access to the authenticator, that is, for $1 \le i \le \mathrm{poly}(k)$, where $\mathrm{poly}(\cdot)$ 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, $\mathcal{A}$ cannot issue more than one query with the same group identification tag; that is, for all $i \neq j$, $GID_i \neq GID_j$.
(3) At the end, $\mathcal{A}$ outputs a tag GID, values n, α and β, and a set RP of packets.

The adversary wins the game if one of the following violations occurs.

Violation of the (α, β)*-correctness property*. The adversary did manage to construct RP such that even if it contains $\alpha_i n_i$ packets of some authenticated packet set AP_i for group identification tag $GID_i = GID$, the decoder, on input GID,

[5]For an (α, β)-network neither can be true, and the decoder will always produce output packets.

n, α, β, and RP, 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 $GID_i = GID$, $n_i = n, \alpha_i = \alpha$, and $\beta_i = \beta$. 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 are included in the received packets, that is, $|RP \cap AP| \geq \alpha n$, and the received packets are at most βn, that is, $|RP| \leq \beta n$.

—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 is received, that is, $a_j \in RP \cap AP$, and yet is 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 manage to construct RP such that the decoder, on input GID, n, α, β, and RP, computed output packets $OP = \{p'_1, \ldots, p'_n\}$ that were never authenticated by the authenticator for the group identification tag GID. Namely, the adversary wins if one of the following holds:

—The authenticator was never queried with group identification tag GID and the size n, and yet the decoder does not reject. That is, it holds that $\mathsf{reject} \neq \mathsf{Decode}(PK, GID, n, \alpha, \beta, RP) = OP$.

—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, the decoder does not reject and either $n \neq n'$ or some output packet $p'_j \neq \varnothing$ is different from the corresponding data packet p_j, where $OP = \{p'_1, \ldots, p'_n\}$.

In practice, n, α, and β are system-wide parameters and GID is chosen uniquely by the sender. In the previous definition, security is guaranteed whenever a new GID is used, but security holds even if the assumption on the (α, β)-network is not met.

4. OUR MULTICAST AUTHENTICATION SCHEME AND AUTHENTICATED ECC

We now describe a multicast authentication scheme (KeyGen, Auth, Decode) that meets the definitions of the previous section. In the sequel, we denote with ϵ, $0 < \epsilon < 1$, the tolerance parameter of the decoder, which controls a trade-off between the error-tolerance ability of the decoder and its performance. Both the authenticator and the decoder know and use the value of this parameter. Recall that this parameter expresses how far away from the ultimate bound for efficient list-decodability the encoding is performed. The higher the ϵ, the further away from the bound the encoding is performed; thus, the communication overhead is higher but the list decoder operates faster (up to constant factors). Similarly, values of ϵ that are closer to zero reduce the communication overhead at the cost of increasing (by constant factors) the decoding time. We will discuss in detail this trade-off when we will perform the efficiency analysis. By $\circ$, we denote concatenation and by $\varnothing$ we appropriately denote either a packet that is

Algorithm 1. Key Generation KeyGen

Input: Security parameter 1^k, signature scheme $(G(\cdot), \mathsf{Sign}_{(\cdot)}(\cdot), \mathsf{Verify}_{(\cdot)}(\cdot,\cdot))$, and collision-resistant hash-function family $\mathcal{H}$.

Output: Secret key SK and public key PK.

(1) Let $(PK_s, SK_s) \leftarrow G(1^k)$ and $H \leftarrow \mathcal{H}(1^k)$.
(2) Set $PK = (PK_s, H)$ and $SK = SK_s$.

empty or the empty string. To avoid notational overload, we often omit the floor and ceiling notation.

4.1 Key Generation and Authenticator

The key-generation algorithm KeyGen receives as input the security parameter 1^k, a signature scheme $(G(\cdot), \mathsf{Sign}_{(\cdot)}(\cdot), \mathsf{Verify}_{(\cdot)}(\cdot,\cdot))$ and a family $\mathcal{H}$ of collision-resistant hash functions (Definitions 2.1.1 and 2.1.2), and operates simply by initializing a signature scheme and a collision-resistant function using the security parameter. Assuming that $(PK_s, SK_s) \leftarrow G(1^k)$ and $H \leftarrow \mathcal{H}(1^k)$, KeyGen sets $PK = (PK_s, H)$ and $SK = SK_s$, that is, $((PK_s, H), SK_s) \leftarrow \mathsf{KeyGen}(1^k)$ (see Algorithm 1).

We now describe the authenticator Auth that uses the tolerance parameter ϵ of the decoder (see Algorithm 2). The idea is as follows. The data packets are each hashed using the collision-resistant hash function and these n packet hashes $h_1, \ldots, h_n$ (along with the group identifier, α and β) are digitally signed to produce signature σ. 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 (is able to reconstruct) all the authentication information. To that end, we encode S, using an $[n, \rho n+1]_q$ Reed-Solomon code in a manner that is tolerant to packet losses and insertions, subject to the network parameters. The encoded authentication information is then appropriately dispersed and appended in the data packets to form the authenticated packets.

In Algorithm 2 (but also in Algorithms 3 and 4), we assume that data packets, signatures, *GID* s, and the values α and β each has a fixed (and publicly known) length so that the use of concatenation $\circ$ introduces no ambiguities. Alternatively, we can assume that there exists a standard (and publicly known) way of delimiting the beginning and end of the bit strings of the previously mentioned data items. Also, in all algorithms, if ρn is not an integer, we use $\lfloor \rho n \rfloor$ in all occurrences of ρn. We note that q is in fact a function of n, α, and β, thus, it does not need be transmitted to the receiver (observe that $|S|$ is a function of n). Also, we assume that the value of ϵ is known also to the encoder; thus, in fact, $C(S) = C_\epsilon(S)$. Finally, we implicitly assume that the size of the problem n along with the parameters of the network α and β are such that $\rho n + 1 < n$, or equivalently, $\frac{\alpha^2}{(1+\epsilon)\beta} + \frac{1}{n} < 1$, so that the used $[n, \rho n + 1]_q$ Reed-Solomon code does not degenerate. This last technical requirement is easily satisfied as n and β get larger and α gets smaller.

Algorithm 2. Authenticator Auth

Input: Secret key SK, public key PK, group identification tag GID, data-stream size n, parameters α and β of the network, and data packets $DP = \{p_1, \ldots, p_n\}$.
Output: Authenticated packets $AP = \{a_1, \ldots, a_n\}$.

(1) For $1 \leq i \leq n$, compute the hash value $h_i = H(p_i)$. The concatenation of all the hash values, together with the value GID, is digitally signed:
$$\sigma \leftarrow \mathsf{Sign}_{SK}(GID \circ \alpha \circ \beta \circ h_1 \circ \ldots \circ h_n).$$
(2) Let $[n, \rho n + 1]_q$ be a Reed-Solomon error-correcting code, where q is defined in Step 3, and set its rate to be $\rho = \frac{\alpha^2}{(1+\epsilon)\beta}$, where α and β are the survival and flood rates of the network, and ϵ is the tolerance parameter of the decoder. (Note that: $\rho < 1$ since $\alpha \leq 1$, $\beta \geq 1$ and $0 < \epsilon < 1$.)
(3) Set $S = h_1 \circ \ldots \circ h_n \circ \sigma$ as the authentication information. Split S into $\rho n + 1$ substrings of size $\left\lceil \frac{|S|}{\rho n+1} \right\rceil$, where the ith substring is viewed as value y_i of $\mathbb{F}_q$, with $q = 2^{\left\lceil \frac{|S|}{\rho n+1} \right\rceil}$. If $|S|$ is not an exact multiple of $\rho n+1$, pad S with ℓ 0's, such that $|S \circ 0^\ell| \bmod (\rho n + 1) \equiv 0$.
(4) Use the resulting set $\{y_1, \ldots, y_{\rho n+1}\}$ of field elements to form $\{(g_i, y_i)|1, \ldots, \rho n + 1\}$ and treat this set of points as an input to the Reed-Solomon encoder (see Definition 2.2.1). Compute the corresponding codeword $C(S)$ using the $[n, \rho n + 1]_q$ Reed-Solomon code (of Step 2). $C(S)$ consists of n elements of $\mathbb{F}_q$, denoted as $(s_1, \ldots, s_n)$.
(5) Let $AP = \{a_1, \ldots, a_n\}$, where for $1 \leq i \leq n$, we have $a_i = GID \circ i \circ p_i \circ s_i$.

4.2 Decoder

Our decoder Decode uses a modification of the GS-Decoder (see Definition 2.2.1 and Theorem 2.2.2) 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 in $\mathbb{F}_q[x]$ of degree at most k such that every $p \in L$ has the property that for at least $t = \sqrt{(1+\epsilon)kn}$ of the i's, $p(x_i) = y_i$. We write $L \leftarrow \mathsf{GSDecode}_\epsilon(n, k, t, \{(x_i, y_i)|1 \leq i \leq n\})$. The modified decoder is specified by parameters that are slightly different: It takes as input up to βn points (x_i, y_i) (some with overlapping x-coordinates) and finds a list of candidate input words, that is, set of points that can be encoded by polynomials in $\mathbb{F}_q[x]$ of degree at most ρn (with $\rho = \frac{\alpha^2}{(1+\epsilon)\beta}$) such that each polynomial agrees with at least αn of the decoder input points (see Corollary 2.2.4).

What is important is that the modified GS-Decoder operates even in the presence of an adversarially chosen set of (x_i, y_i) pairs. In other words, the modified decoder corresponds to the alphabet and encoder of the Reed-Solomon code described in Definition 2.2.1 (the encoder is used in algorithm Auth before), but now, the set of points (x_i, y_i) that constitute the input of the decoder is in principle different than the set of valid output points of the encoder; this corresponds to the various attacks by the adversary. For instance, this set may not include some of the original points, may be larger because some new points are added, and may even contain points that are vertically aligned, that is, some of the x_i's are not distinct. The modified decoder is obtained by adapting the original GS-Decoder, as follows (see Algorithm 3).

Algorithm 3. Modified GS-Decoder $\mathsf{MGSDecoder}_\epsilon$

Input: n, α, β, and m points (x_i, y_i), $x_i \in \mathbb{F}_q$, $y_i \in \mathbb{F}_q$, $1 \le i \le m$.
Output: List of all computed candidate input words $\{c_1, \ldots, c_\ell\}$ or reject.

(1) If $m > \beta n$, reject; else, if there are fewer than αn distinct values of x_i, reject.
(2) Else, 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; else, $L = \{Q_1(x), \ldots, Q_\ell(x)\}$ contains polynomials in $\mathbb{F}_q[x]$.
(3) For each $Q_j(x) \in L$, $1 \le j \le \ell$, evaluate $Q_j(x)$ on $g_i \in G_q$, $1 \le i \le \rho n + 1$, and let the string $Q_j(g_1) \circ Q_j(g_2) \circ \cdots \circ Q_j(g_{\rho n+1})$ be candidate input word c_j.

For this decoder, operating on input points subject to constraints that are in accordance with our (α, β)-network, we have the following.

LEMMA 3.2.1. *If at least αn out of at most βn input points are valid, then* $\mathsf{MGSDecoder}_\epsilon$ *does not reject, 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 input words that are consistent with αn of the received points.*

PROOF. $\mathsf{MGSDecoder}_\epsilon$ does not reject in Step 1 of the algorithm. All claims follow considering Step 2, Theorem 2.2.2, Corollary 2.2.4, and the fact that GS-Decoder operates even when the x_i's are not distinct (see Guruswami and Sudan [1999] and Guruswami [2001]). Corollary 2.2.4 holds, since, if $m = \gamma n$, $\alpha \le \gamma \le \beta$, then $\rho n = \frac{\rho}{\gamma} m$; thus, we consider the polynomial reconstruction problem on inputs $\frac{\rho}{\gamma} m$, t, m points, and for $\rho = \frac{\alpha^2}{(1+\epsilon)\beta}$, we have that $t \ge \alpha n \ge \sqrt{\frac{\gamma}{\beta}} \alpha n = \sqrt{(1+\epsilon)\rho n m} = \sqrt{(1+\epsilon)\frac{\rho}{\gamma}} m$, as needed. Of course, $\tilde{O}(m^2) = \tilde{O}(n^2)$ for $m = \gamma n$ and γ a constant. Finally, we have that $\sqrt{(1+\epsilon)\frac{\rho}{\gamma}} = \sqrt{\frac{\alpha^2}{\beta\gamma}} \le 1$ as required. The correct answer (candidate input word) is guaranteed to be contained in the output list, since it corresponds to (or produced by) a polynomial of degree $\le \rho n$ that is consistent with αn points; thus, $\mathsf{MGSDecoder}_\epsilon$ does not reject in Step 2 either. The arguments hold also in the case where $\lfloor \rho n \rfloor$ is considered instead of ρn. □

Now, we are ready to describe our decoder. The idea is simple: We resist every attack by the adversary by treating injected, altered, and lost packets as errors, essentially, in a crypto-enhanced list-decoder (see Algorithm 4). What matters is only the achieved agreement, that is, the valid packets. Note that the decoder is parameterized by the tolerance parameter ϵ (i.e., Decode is in fact Decode_ϵ).

We postpone the analysis of the running time of the algorithms KeyGen, Auth, and Decode of our scheme until the next section.

4.3 Correctness and Security Proofs

For the decoder in Algorithm 4, we have the following:

LEMMA 4.3.1. *When operating on a stream of packets encoded by authenticator* Auth *and transmitted through an (α, β)-network, algorithm* Decode *does not reject.*

Algorithm 4. Decoder Decode

Input: Public key PK, group identification tag GID, n, parameters α and β, and received packets $RP = \{r_1, \ldots, r_m\}$.
Output: $OP = \{p'_1, \ldots, p'_n\}$ or reject.

(1) View packets in RP as $r_i = GID_i \circ j_i \circ p_i \circ s_i$.
(2) Discard all nonconforming packets, that is, all packets for which $GID_i \neq GID$ 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 $r_i = GID \circ j_i \circ p_i \circ s_i$, such that $j_i \in [1..n]$.
(3) If $m' < \alpha n$ or $m' > \beta n$, then reject.
(4) For $1 \leq i \leq m'$, 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), $1 \leq i \leq m'$. If $\mathsf{MGSDecoder}_\epsilon$ rejects, reject; otherwise, obtain the candidate input words $\{c_1, \ldots, c_\ell\}$.
(6) For $1 \leq i \leq n$, set $h_i = \varnothing$. Let $j = 1$. While $j \leq \ell$:
    —Parse the candidate input word c_j as string $h_1^j \circ \ldots \circ h_n^j \circ \sigma$.
    —If $\mathsf{Verify}_{PK_s}(GID \circ \alpha \circ \beta \circ h_1^j \circ \ldots \circ h_n^j, \sigma) = 1$, then set $h_i = h_i^j$ for $1 \leq i \leq n$ and break out of the loop; otherwise, increment j.
(7) If $(h_1, \ldots, h_n) = (\varnothing, \ldots, \varnothing)$, reject; else, compute 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 $r_i = GID \circ j \circ p_j \circ s_j$, $j \in [1..n]$, and if $H(p_j) = h_j$, set $p'_j = p_j$.

PROOF. From the properties of the (α, β)-network, the algorithm does not reject in Steps 3 and 5. Since at least αn packets are valid, from Lemma 4.2.1, we have that the correct input word (which stores the entire authentication information S) is among the candidates of the output list of Step 5. Thus, the corresponding signature verification in Step 6 excludes the rejection in Step 7. □

Let us show that the authentication scheme $(\mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Decode})$ described in Sections 4.1 and 4.2 satisfies Definition 3.2.1. Suppose that we have an adversary $\mathcal{A}$ who manages to break the (α, β)-correctness or security of our scheme with (nonnegligible) probability $\pi(k)$. Then, one of the following is true.

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

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

CLAIM 4.3.2. *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. We exhibit a reduction that 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_s of the signature scheme. Our reduction is also given oracle access to the corresponding signer $\mathsf{Sign}_{SK_s}(= \mathsf{Sign}_{SK})$. The reduction sets up the public key $PK = (PK_s, H)$. 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 $(GID_i, n_i, \alpha_i, \beta_i, DP_i)$, run the algorithm Auth with the following modification: In Step 1, at the beginning of 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. That is, it outputs values GID, n, α, β, and the set of received packets RP such that all of the following hold:

(1) $GID = GID_i, n = n_i, \alpha = \alpha_i$, 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, $\sigma_i \leftarrow \mathsf{Sign}_{SK}(GID \circ \alpha \circ \beta \circ H(p_1) \circ \ldots \circ H(p_n))$.
(2) $|RP \cap AP| \geq \alpha n$ and $|RP| \leq \beta n$.
(3) For some j, $p_j \neq p'_j$, where $(p'_1, \ldots, p'_n) \leftarrow \mathsf{Decode}(PK, GID, n, \alpha, \beta, RP)$, and $r_j \in RP$ is a packet that corresponds to packet $p_j \in DP_i$. (Note that from 2 and Lemma 4.3.1, it follows that algorithm Decode does not reject.)

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, as $p_j \neq p'_j$ and using a straightforward reduction. By definition of Decode, if $r_j \in RP$ and $p'_j \neq \varnothing$, then, in Step 6, the algorithm Decode processes a candidate $c = h_1 \circ \ldots \circ h_n \circ \sigma$ such that $\mathsf{Verify}_{PK_s}(GID \circ \alpha \circ \beta \circ h_1 \circ \ldots \circ h_n, \sigma) = 1$. We must argue that our signature oracle was never queried on input $(GID \circ \alpha \circ \beta \circ h_1 \circ \ldots \circ h_n)$. Note that the only time it was queried with this GID was when we obtained σ_i on input $(GID \circ \alpha \circ \beta \circ H(p_1) \circ \ldots \circ H(p_n))$. Moreover, in Step 7, Decode includes p'_j into OP if and only if $H(p'_j) = h_j$. Therefore, $h_j \neq H(p_j)$, so our signature oracle was never queried with $(GID \circ \alpha \circ \beta \circ h_1 \circ \ldots \circ h_n)$, and yet our adversary has caused us to compute a signature σ such that $\mathsf{Verify}_{PK_s}(GID \circ \alpha \circ \beta \circ h_1 \circ \ldots \circ h_n, \sigma) = 1$. 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 4.2.1), 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 $\mathsf{Verify}_{PK_s}(GID \circ \alpha \circ \beta \circ h_1 \circ \ldots \circ h_n, \sigma) = 1$. But, the only query with GID

that we ever issued to the signer was for the message $(GID \circ \alpha \circ \beta \circ H(p_1) \circ \ldots \circ H(p_n)) \neq (GID \circ \alpha \circ \beta \circ h_1 \circ \ldots \circ h_n)$. Thus, σ is a successful forgery. □

CLAIM 4.3.3. *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 4.3.2. 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, size n, α, and β and yet the decoder algorithm does not reject. That is, $\mathsf{reject} \neq \mathsf{Decode}(PK, GID, n, \alpha, \beta, RP) = OP$.
(2) The authenticator was queried with the group identification tag GID, with the values n', α', and β', and with data packets $DP = \{p_1, \ldots, p_{n'}\}$. However, the decoder algorithm does not reject and 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, we know that the only way the decoder will produce some nonempty set of output packets is if, in Step 6, it sees a string c and a signature σ such that $\mathsf{Verify}_{PK_s}(GID \circ c, \sigma) = 1$. Since the signature oracle was never queried for this GID, α, β, and n, σ is a successful forgery. So, suppose that (2) holds. This situation is identical to Case 1 of the proof of Claim 4.3.2, and we obtain either a successful forgery or a hash-function collision in exactly the same manner. □

4.4 Authenticated Reed-Solomon Error-Correcting Code

Our authentication scheme consists of the PPT algorithms (KeyGen, Auth, Decode), described in Subsections 4.1 and 4.2. We note that essentially our scheme uses an authenticated Reed-Solomon error-correcting code. Using this term, we refer to the fact that, by allowing the use of cryptographic primitives in the data that is encoded, list decoding succeeds in operating (even) in the presence of adversarial behavior of the "transmission channel" (a network in our case). In general, depending on this adversarial behavior, our authenticated decoder either rejects or correctly and securely reconstructs the original codeword that was sent and the input word that was produced by the encoder (authentication information in our case). For an (α, β)-network, Lemma 4.3.1 and Claim 4.3.2 guarantee that the correct reconstruction is produced and Claim 4.3.3 guarantees that the authenticated Reed-Solomon error-correcting code is secure. For this reason, we refer to our multicast authentication scheme as AuthECC.

Definition 4.4.1 (*Authenticated ECC*). Multicast authentication scheme AuthECC comprises the triplet of probabilistic algorithms (KeyGen, Auth, Decode) described earlier in the text and realizes an authenticated Reed-Solomon error-correcting code.

Our AuthECC constitutes a general-purpose authentication tool for data streams or other unstructured data formats and, as we have proved, satisfies the following.

THEOREM 4.4.2. *Multicast authentication scheme* AuthECC *is* (α, β)*-correct and secure for any* (α, β)*-network.*

Finally, from the previous result, we immediately get the following corollary, which draws an interesting connection between coding theory and cryptography.

COROLLARY 4.4.3. *In a public-key setting and for computationally bounded communication channels, list decoding can be transformed into unambiguous decoding.*

5. ANALYSIS

We now analyze our scheme in terms of the various cost parameters. Recall that α is the survival rate of the network, where $0 < \alpha \leq 1$, β is the flood rate of the network, where $\beta \geq 1$, ϵ is the tolerance parameter of the list decoder, where $0 < \epsilon < 1$ and ρ is the rate of the encoder, and where $\rho = \frac{\alpha^2}{(1+\epsilon)\beta}$ and $\rho < 1$.

We start by discussing the complexity of our authentication scheme AuthECC in terms of computational and communication costs and introduced delay, we then examine how our scheme can be tuned and extended, and finally, we finish the section by comparing our scheme with various previous proposed schemes. In the sequel, by h, we denote the size of a hash value, and by s, we denote the size of a digital signature.

Computational Cost. The sender and the receiver execute algorithms Auth and Decode, respectively. Both algorithms involve field operations, namely 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 of these operations take $O(\frac{h}{\rho} \log^{O(1)} \frac{h}{\rho})$ time [Guruswami 2001]. 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, the authentication information is encoded using the systematic version of the Reed-Solomon code, a task that requires $O(n \log n)$ field operations or $O(n \log n\ N \log^{O(1)} N)$ time. To see where these bounds come from, recall that the Reed-Solomon encoding algorithm is just polynomial interpolation and evaluation; the standard algorithms for interpolation and evaluation use the fast Fourier transform and take $O(n \log n)$ arithmetic operations assuming that the canonical set of points on which the polynomial is evaluated was chosen appropriately (see Cormen et al. [2001] for an overview of the relevant algorithms and for references to efficient implementations). The version of the Reed-Solomon code that we use is a systematic code.

—*Decoder*. From Theorem 2.2.2 and Lemma 4.2.1, 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 ($\tilde{O}(\cdot)$-notation omits some logarithmic

factors). 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 in $O(n \log nN \log^{O(1)} N)$ 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 $\frac{n}{\rho n+1}(s + hn)$. That is, we have constant communication overhead per packet $\frac{s+hn}{\rho n+1} < \frac{h}{\rho} + \frac{s}{\rho n} = \frac{h}{\rho} + o(1)$. 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. Delay is defined as the number of packets that the authenticator or decoder algorithm needs to buffer. We observe that, by definition, any authentication scheme in our model has $O(n)$ delay, as it needs to process $O(n)$ packets. Delay is an important cost parameter, since it captures the ability of the authenticator or the decoder to process packets in a online fashion. In our scheme, the sender processes n packets, and the receiver processes βn packets in the worst case. However, the receiver can attempt an early decoding by invoking a (different) decoding procedure after only $\rho n + 1$ or αn packets have been received. Details on early decoding follow.

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, thus, the first $\rho n + 1$ symbols of $E(S)$ equal S (where S is the authentication information). Of course, to reconstruct S, we need that no packet loss has occurred among these packets and that the packets arrive in their correct transmitted order. If the first $\rho n + 1$ packets carry information that form a candidate codeword $S = h_1 \circ \ldots \circ h_n \circ \sigma$ and if signature σ is verified, then S is the correct authentication information that 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 less expensive (since the decoder process fewer packets, the minimum required number for correct decoding if the network exhibits no malicious behavior), and if the correct authentication information is computed, no attack is in process and the delay is αn. In this case, packet rearrangements (due to out-of-order incoming packets) are tolerated. 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.

Related to the introduced delay at the receiver, we note that in a realistic scenario where a long data stream is transmitted over a network, the sender will most likely partition the stream into a number of groups, each of size n, and will apply the authenticator separately on each such group, each time using a different per-group *GID* (unique for the entire session of transmissions). Depending on the overlap in their transmission time, packets originated from

Table I. Per-packet Communication Cost c for Various Values of the Survival Rate α, flood rate β, and Tolerance Parameter ϵ

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
0.50	1.5	0.1	6.60	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	4.40	88
0.50	1.5	0.5	9.00	180	0.5	2	0.1	8.80	176
0.75	1.5	0.5	4.00	80	0.5	3	0.1	13.20	264

We assume the use of the SHA-1 hashing algorithm, that is, $h = 20$ bytes. Cost c should be compared with the size s of the signature in use (e.g., $s = 256$ bytes for RSA signatures). Recall that $\rho = \frac{\alpha^2}{(1+\epsilon)\beta}$ is the rate of the code in use and that $c = \frac{h}{\rho} = \frac{\beta(1+\epsilon)}{\alpha^2}h$.

different groups may reach the receiver simultaneously. Recall that each authenticated packet carries its corresponding *GID* and index in the group it is contained. By using this information, the receiver is, therefore, able to separate incoming packets according to their group and also place them in the appropriate position within each group. Thus, in practice, intermixed groups may increase the processing delay at the receiver; such extra delays can, however, be limited by appropriate use of time outs.

Based on the previous text, the performance of our scheme is summarized as follows:

THEOREM 5.0.4. *For authenticating n packets in any* (α, β)*-network, multicast authentication scheme* AuthECC *achieves the following performance, where* ϵ *is the tolerance parameter of the decoder: (i) the sender performs one signature operation, n hash computations and* $O(n \log n)$ *field operations; (ii) the receiver performs* $O(1)$ *signature verifications,* βn *hash computations and* $\tilde{O}(n^2)$ *field operations; (iii) the communication overhead is constant per packet, proportional to* $\frac{\beta}{\alpha^2}$*; (iv) the delay at the sender is n packets; and (v) the delay at the receiver is at most* βn *packets for tolerating attacks, and only* $\frac{\alpha^2}{(1+\epsilon)\beta}n + 1$ *or* αn *packets for detecting attacks.*

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 I shows the communication overhead per packet for specific values of α, β, and ϵ. Here, we note that the decoding cost of our scheme is proportional to β^2.

Beyond parameter set-up, 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 by Pannetrat and Molva [2003]. 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 [Pannetrat and Molva 2003], 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 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 $\{H, H'\} \leftarrow C[\gamma n, n]_{q_1}(H)$ and then $\{A, A'\} \leftarrow C[n, \rho n + 1]_{q_2}(A)$, where $A = \sigma \circ H' \circ \sigma'$, $\sigma' = \mathsf{Sign}_{SK}(\sigma \circ H')$, $\sigma = \mathsf{Sign}_{SK}(H)$, and $\gamma = 1 - \alpha + \sqrt{(1+\epsilon)\beta}$, $q_1 = 2^h$, $\rho = \frac{\alpha^2}{(1+\epsilon)\beta}$, $q_2 = 2^{\lceil \frac{|A|}{\rho n+1} \rceil}$. As in our basic scheme, $A \circ A'$ is split in n equal shares A_i, and packet p_i corresponds to authenticated packet $a_i = GID \circ i \circ p_i \circ A_i$. Given that at the decoder at least αn packets will be valid, it is guaranteed that a constant-size list of candidate strings for $\sigma \circ H' \circ \sigma'$ can be produced by first list-decoding according to code $[n, \rho n + 1]_{q_2}$; then also, list decoding can be transformed to unambiguous decoding by verifying a constant number of signatures σ' on candidate strings $\sigma \circ H'$. Thus, the decoder can compute $\sigma \circ H'$, where H' corresponds to $(\gamma - 1)n$ points. Furthermore, the receiver is always capable to next list decode according to code $[\gamma n, n + 1]_{q_1}$ to eventually get the packet hashes H (after verifying a constant number of signatures σ on candidate strings H). Indeed, if δn in total packets reach the receiver, $\delta \leq \beta$, then the agreement condition of Corollary 2.2.4 holds, since $t \geq \alpha n + (\gamma - 1)n \geq \sqrt{(1+\epsilon)\delta n}$. The new per-packet communication overhead is $\frac{(\gamma-1)h}{\rho}$, thus improved for $\gamma < 2$. Therefore, for any α and β so that $\beta < \frac{(\alpha+1)^2}{1+\epsilon}$, the communication cost is decreased by a constant factor of γ; the computational cost is roughly doubled, since two encodings are performed.

A second modification can be performed by decreasing the field size to appropriately reduce the cost of 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 of γ.

5.2 Comparison with Other Schemes

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

Sign-All and Merkle Tree Schemes. These schemes are resilient to fully adversarial networks. The sign-all scheme involves one signature (verification, respectively) 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 the communication and the computational costs of our scheme can be compared to the corresponding costs of the sign-all scheme. We next present a qualitative comparison between the sign-all scheme and our scheme.

Very short signature schemes have recently been proposed [Boneh et al. 2001]. While the length of a signature can be as low as 160 bits, the security of this scheme is only proven in the random oracle model, and only under a strong assumption (Diffie-Hellman assumption in gap-DH groups [Boneh and Franklin 2003]). 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 scheme, such as the one by Cramer and Shoup [2000] or its modification due to Fischlin [2003], will add about 500 bytes to each packet—which is more than what we have for reasonable α and β.

With respect to the computational cost, the n hash operations and $O(n \log n)$ field operations executed by the sender in our scheme are likely to be less expensive than the n signatures of the sign-all scheme. On the other hand, the $\tilde{O}(n^2)$ field operations performed by the receiver in our scheme may or may not be faster than the n signature verifications of the sign-all scheme, depending on constant factors, the signature scheme in use, and the specific value of n.

Additionally, signing every packet is undesirable in practice. By signing every packet separately, we lose both in efficiency and in architecture design, since the secret key operations are computationally expensive and need extra 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 or, as identified and described in Karlof et al. [2004], signature flooding 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 [Wong and Lam 1999] has better time complexity than our scheme. For a group of packets of size n, only $2n$ hash computations and one signature computation (verification, respectively) are performed at the sender (receiver, respectively). However, the Merkle-tree scheme has communication cost that grows with the number of packets, thus is theoretically less scalable, though in practice, this logarithmic per-packet communication overhead may be acceptable. Our scheme is theoretically optimal in terms of communication cost: packets have constant authentication overhead.

Operating in a fully adversarial network, the Merkle-tree scheme has also the drawback that is vulnerable to signature flooding attacks. Of course, by appropriately caching hash values, we can significantly resist against this attack: Once the first valid packet is verified, its (authenticated) hashes are stored, then subsequent packets need be verified only with respect to the hashes—and not the signature—they carry. Thus, injected packets that are received afterward do not cause signature verifications. Although this kind of attack is thus terminated after the first correct signature verification, still, in a fully adversarial network with packets rearrangements, injected packets will precede the valid ones. In our (α, β)-network model, the Merkle-tree scheme needs $(\beta - \alpha)n$ signature verifications. Instead, our scheme performs only a constant number of signature verifications at the receiver.

Graph-Based Schemes. These schemes 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 scheme with $O(1)$ per-packet communication overhead will have the undesirable property that $O(1)$ critical packets can be adversarially chosen to disconnect the signature node (packet) from the authentication chain. In the piggybacking scheme in Miner and Staddon [2001], this number of critical packets can be $O(n)$ at the expense of $O(n)$ per-packet communication overhead. Our scheme does not have these drawbacks, since the signature is dispersed among all packets. Unlike 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 totally symmetric: All packets share the authentication information.

Erasure-Code Schemes. The first two erasure-code schemes [Pannetrat and Molva 2003; Park et al. 2003] 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 cause the decoding procedure to fail. For networks where packets get only lost, they perform slightly better than our scheme in terms of computational and communication costs. This is due to the fact that erasure codes are more time/space-efficient than error-correcting codes. 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 our scheme, tolerating injected packets comes at this small price of having slightly worse performance than erasure-code schemes.

Table II. Comparison of Selected Multicast Authentication Approaches, No (α, β)-Correct and Secure, with Respect to Various Aspects of Efficiency, Security, and Resiliency

	Sign-all	GB I	GB II	Erasure	AuthECC
Delay (Sender)	1	n	n	n	n
Computation (Sender)					
Sign	n	1	1	1	1
hash	—	$O(n)$	$O(n^2)$	n	n
field op	—	—	—	$O(n \log n)$	$O(n \log n)$
Communication	sn	$O(hn)$	$O(hn^2)$	$\frac{1-\alpha}{\alpha} hn$	$\frac{\beta(1+\epsilon)}{\alpha^2} hn$
Delay (Receiver)	1	n	n	n	βn
Computation (Receiver)					
Verify	n	1	1	1	$O(1)$
hash	—	$O(n)$	$O(n^2)$	n	βn
field op	—	—	—	$O(n^2)$	$\tilde{O}(n^2)$
Secret key protection	—	•	•	•	•
Resiliency					
Chosen packet loss	•	—	•	•	•
Chosen packet injection	•	•	•	—	•
Signature dispersal	•	—	—	•	•

Here, Sign denotes a signature operation, Verify a signature verification, hash the total hashing cost (where the cost of hashing a string is considered to be a linear function of the input size), s the signature size, and h the hash size. The communication and computational costs refer to a group of n packets.

Using distillation codes, Karlof et al. [2004] tolerate packet injections, but their proposed scheme has high communication overhead and is, thus, less scalable, because a Merkle hash tree is used to "filter out" the injected packets (and, thus, the communication cost is $O(\log n)$). For such a logarithmic communication overhead, the scheme by Wong and Lam [1999] may be actually preferable, since it has both lower time complexity and better resiliency to adversarial network behavior. In terms of computational effort at the sender and receiver, this scheme is similar to our scheme except from the following two points regarding the computational effort at the receiver. In Karlof et al. [2004], partitioning the packets into groups introduces an extra computational overhead and the total number of hash values computed is by a logarithmic factor larger. In our scheme, the constants involved in the quadratic decoding process are higher than the ones in Karlof et al. [2004]. Finally, the model used in Gunter et al. [2004] tolerates no adversarial packet losses.

Other Schemes. TESLA by Perrig et al. [2000, 2001] and the scheme by Xu and Sandhu [2002] 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.

Tables II and III summarize the previous discussion, where selected schemes are compared with our scheme AuthECC. Table II compares our scheme with the sign-all solution and various selected schemes that are not (α, β)-correct and secure. We consider two graph-based authentication schemes, one of constant degree (expander construction [Song et al. 2002], denoted as GB I) and one of $O(n)$ degree (piggybacking scheme with parameterized performance [Miner and Staddon 2001], denoted as GB II, where we assume a constant number

Table III. Comparison of the Three (α, β)-Correct and Secure Multicast Authentication Schemes with Respect to Various Aspects of Efficiency, Security and Resiliency, and for a Group of n Packets

	Merkle	Distillation Code	AuthECC
Delay (Sender)	n	n	n
Computation (Sender)			
Sign	1	1	1
hash	$2n$	$3n$	n
field op	—	$O(n \log n)$	$O(n \log n)$
Communication	$(s + h \log n)n$	$(\frac{1}{\alpha} + \log n)hn$	$\frac{\beta(1+\epsilon)}{\alpha^2}hn$
Delay (Receiver)	1	βn	βn
Computation (Receiver)			
Verify	$(\beta - \alpha)n$	$\frac{\beta}{\alpha}$	$O(1)$
hash	$2n$	βn	βn
field op	—	$O(n^2)$	$O(n^2)$
Secret key protection	•	•	•
Resiliency			
Chosen packet loss	•	•	•
Chosen packet injection	•	•	•
Signature dispersal	•	•	•

of classes), and one erasure scheme (optimized in terms of communication scheme [Pannetrat and Molva 2003]). Table III compares our scheme with the only two (α, β)-correct and secure previous approaches, namely the schemes by Wong and Lam [1999] and Karlof et al. [2004].

6. CONCLUSION

In this article, we propose a new general framework for the multicast authentication problem, where the network is controlled by a computationally bounded adversary that has great power in modifying the transmitted stream. Our model is realistic in terms the of adversarial behavior. The limitations on the adversary's power, characterized by the survival and flood rates, exclude from consideration only degenerate cases, where the authentication problem becomes meaningless.

Our work establishes a new direction in data-stream authentication by going beyond erroneous networks and addressing fully adversarial networks. Based on a novel combination of primitives from coding theory and cryptography, our solution realizes an authenticated error-correcting code that constitutes a new authentication tool, which can be useful in other settings as well. Our scheme is efficient 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 other schemes [Karlof et al. 2004; Wong and Lam 1999]. When compared with the Merkle-tree scheme, the $O(n^2)$ time complexity of our scheme at the receiver 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, it is worth investigating the practical performance of our scheme through an implementation and experimental study. Also, a natural question to explore is whether the decoding procedure can be simplified and whether the time complexity can be improved. One other question is whether other classes of error-correcting codes can provide further benefits in runtime or communication overhead. Moreover, in this article, we showed a connection between coding theory and cryptography by employing cryptographic primitives to unambiguously list decode an error-correcting code. Studying additional topics in this direction seems very interesting. Finally, it is interesting to explore the use of our technique in other data authentication problems.

ACKNOWLEDGMENTS

We would like to thank Philip Klein for useful discussions and also the anonymous reviewers of the 2004 IEEE Symposium of Security & Privacy and of ACM TISSEC for their useful and constructive comments.

REFERENCES

BONEH, D., DURFEE, G., AND FRANKLIN, M. 2001. Lower bounds for multicast message authentication. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'01)*. Springer, Berlin, 437–452.

BONEH, D. AND FRANKLIN, M. 2003. Identity-based encryption from the Weil pairing. *SIAM J. Comput. 32*, 3, 586–615.

BONEH, D., LYNN, B., AND SHACHAM, H. 2001. Short signatures from the Weil pairing. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'01)*. Springer, Berlin, 514–532.

CANETTI, R., GARAY, J., ITKIS, G., MICCIANCIO, D., NAOR, M., AND PINKAS, B. 1999. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of the 27th IEEE International Conference on Computer Communications*(*INFOCOM'99)*. IEEE, Los Alamitos, CA, 708–716.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. 2001. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

CRAMER, R. AND SHOUP, V. 2000. Signature schemes based on the strong RSA assumption. *ACM Trans. Inf. Syst. Secur. 3*, 3, 161–185.

DESMEDT, Y., FRANKEL, Y., AND YUNG, M. 1992. Multi-receiver/multi-sender network security: Efficient authenticated multicast/feedback. In *Proceedings of the 20th IEEE International Conference on Computer Communications* (*INFOCOM'92)*. IEEE, Los Alamitos, CA, 2045–2054.

FISCHLIN, M. 2003. The Cramer-Shoup strong-RSA signature scheme revisited. In *Proceedings of the International Conference on Practice and Theory in Public Key Cryptography*. Springer, Berlin, 116–129.

GENNARO, R. AND ROHATGI, P. 1997. How to sign digital streams. In *Proceedings of the 17th Annual International Cryptology Conference*(*CRYPTO'97)*. Springer, Berlin, 180–197.

GOLDBERG, I. 2007. Improving the robustness of private information retrieval. In *Proceedings of the Symposium on Security & Privacy*. IEEE, Los Alamitos, CA, 131–148.

GOLDREICH, O. 2004. *Foundations of Cryptography*, vol II. Cambridge University Press, Cambridge, MA.

GOLDREICH, O., RUBINFELD, R., AND SUDAN, M. 2000. Learning polynomials with queries: The highly noisy case. *SIAM J. Disc. Math. 13*, 4, 535–570.

GOLDWASSER, S., MICALI, S., AND RIVEST, R. 1988. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput. 17*, 2, 281–308.

GOLLE, P. AND MODADUGU, N. 2001. Authenticating streamed data in the presence of random packet loss. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, Reston, VA, 13–22.

GUNTER, C., KHANNA, S., TAN, K., AND VENKATESH, S. 2004. DoS protection for reliably authenticated broadcast. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, Reston, VA, 17–36.

GURUSWAMI, V. 2001. List decoding of error-correcting codes. Ph.D. thesis, Massachusetts Institute of Technology, Boston, MA.

GURUSWAMI, V. AND SUDAN, M. 1999. Improved decoding of Reed-Solomon and algebraic geometric codes. *IEEE Trans. Inf. Theory 45*, 1757–1767.

KARLOF, C., SASTRY, N., LI, Y., PERRIG, A., AND TYGAR, J. 2004. Distillation codes & applications to DoS resistant multicast authentication. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, Reston, VA, 37–56.

KRAWCZYK, H. 1993. Distributed fingerprints and secure information dispersal. In *Proceedings of the Symposium on Principles of Distributed Computing*. ACM, New York, 207–218.

KROHN, M., FREEDMAN, M., AND MAZIERES, D. 2004. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Proceedings of the Symposium on Security & Privacy*. IEEE, Los Alamitos, CA, 226–240.

LUBY, M. G., MITZENMACHER, M., SHOKROLLAHI, M. A., AND SPIELMAN, D. A. 2001. Efficient erasure correcting codes. *IEEE Trans. Inf. Theory 47*, 2, 569–584.

LYSYANSKAYA, A., TAMASSIA, R., AND TRIANDOPOULOS, N. 2004. Multicast authentication in fully adversarial networks. In *Proceedings of the Symposium on Security & Privacy*. IEEE, Los Alamitos, CA, 241–255.

MCELIECE, R. J. 2003. The Guruswami-Sudan decoding algorithm for Reed-Solomon codes. Tech. rep. JPL Interplanetary Network Progress Report, IPN PR 42–153.

MICALI, S., PEIKERT, C., SUDAN, M., AND WILSON, D. A. 2005. Optimal error correction against computationally bounded noise. In *Proceedings of the Theory of Cryptology Conference (CRYPTO'05)*. Springer, Berlin, 1–16.

MINER, S. AND STADDON, J. 2001. Graph-based authentication of digital streams. In *Proceedings of the Symposium on Security & Privacy*. IEEE, Los Alamitos, CA, 232–246.

PANNETRAT, A. AND MOLVA, R. 2003. Efficient multicast packet authentication. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, Reston, VA.

PARK, J. M., CHONG, E. K. P., AND SIEGEL, H. J. 2003. Efficient multicast packet authentication using erasure codes. *ACM Trans. Inf. Syst. Secur. 6*, 2, 258–285.

PERRIG, A. 2001. The BiBa one-time signature and broadcast authentication protocol. In *Proceedings of the Conference on Computing and Communication Security*. ACM, New York, 28–37.

PERRIG, A., CANETTI, R., SONG, D., AND TYGAR, J. 2001. Efficient and secure source authentication for multicast. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, Reston, VA, 35–46.

PERRIG, A., CANETTI, R., TYGAR, J., AND SONG, D. 2000. Efficient authentication and signing of multicast stream over lossy channels. In *Proceedings of the Symposium on Security & Privacy*. IEEE, Los Alamitos, CA, 56–73.

RABIN, M. O. 1989. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM 36*, 2, 335–348.

REED, I. S. AND SOLOMON, G. 1960. Polynomial codes over certain finite fields. *SIAM J. Appl. Math. 8*, 2, 300–304.

ROHATGI, P. 1999. A compact and fast hybrid signature scheme for multicast packet authentication. In *Proceedings of the Conference on Computing and Communication Security*. ACM, New York, 93–100.

SIMMONS, G. J. 1984. Authentication theory / coding theory. In *Proceedings of the Theory of Cryptology Conference (CRYPTO'84)*. Springer, Berlin, 411–431.

SONG, D., ZUCKERMAN, D., AND TYGAR, J. D. 2002. Expander graphs for digital stream authentication and robust overlay networks. In *Proceedings of the Symposium on Security and Privacy*. IEEE, Los Alamitos, CA, 258–270.

WELCH, L. AND BERLEKAMP, E. 1986. Error correction of algebraic block codes. U.S. Patent Number 4,633,470, issued December 1986.
WONG, C. K. AND LAM, S. S. 1999. Digital signatures for flows and multicasts. *IEEE/ACM Trans. Networking* 7, 4, 502–513.
XU, S. AND SANDHU, R. 2002. Authenticated multicast immune to denial-of-service attack. In *Proceedings of the Symposium on Applied Computing*. ACM, New York, 196–200.

Received April 2006; revised February 2009; accepted July 2009