A Multiple-MAC-Based Protocol to Identify Misbehaving Nodes in Network Coding

Juan Camilo Corena
Graduate School of Science and Technology
Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223-8522, Japan
Email: jcorena@z8.keio.jp

Tomoaki Ohtsuki
Graduate School of Science and Technology
Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223-8522, Japan
Email: ohtsuki@ics.keio.ac.jp

***Abstract*—In network coding, intermediate nodes are allowed to transmit a function of the packets, instead of the traditional scheme where unmodified packets travel through the network. Some of the advantages of this mechanism are: a unified way to represent Broadcast, Multicast and Unicast, robustness in link and node failures and robustness to routing loops. However, allowing intermediate nodes to change information, introduces new points where byzantine attackers may try to disrupt the network. In this paper, we present a Message Authentication Code (MAC) Based protocol which can identify misbehaving nodes. Our construction uses only fast symmetric cryptographic operations, making it suitable for multicast networks, where latency is an important factor. For its construction, we used an efficient key assignment based on Blom's scheme, and a Merkle tree to provide authenticity during our identification routine. We show our construction is relevant in the context of network coding, by showing its execution time compared to that of pollution detection routines and other schemes used for authentication.**

I. Introduction

The problem of detecting misbehavior in network coding has received significant attention in academic literature [1], [2]. The so called "Pollution Attack" where a malicious node injects packets in the network, is one of the problems related to this technology that has received more attention. Despite this, there are still problems that need to be solved in this matter; such as detecting and isolating the attackers efficiently.

One crucial step needed to ban misbehaving nodes from the network, is to provide an efficient way for honest nodes to authenticate other nodes; otherwise, attackers could easily impersonate legitimate nodes to avoid detection; this problem is known as Broadcast Authentication.

In this article we present a mechanism that allows honest nodes to exclude misbehaving nodes, by not forwarding packets coming from these nodes in the future. To achieve detection, we base our scheme on the security properties of systems aimed at detecting pollution in network coding. To provide authentication, we present a fast authentication scheme based on Blom's key distribution scheme [3]. Our construction guarantees either that the sender of a packet is identified or an honest node does not waste significant resources checking a packet whose sender cannot be identified.

Unlike other constructions in the literature [4], [5], where a central authority is needed to exclude a misbehaving node; our construction allows any honest node to take proper action against attackers, without contacting the network's central authority; since it is based on cryptographic primitives with provable security properties, its security is high. Our proposal is also reasonable in terms of assumptions; we do not require knowledge of the topology beyond neighboring nodes, making it suitable for dynamic networks. The price we pay is: linear transmission overhead in the number of neighbors a given node has. Another not so significant drawback is that, the proposal becomes vulnerable when more than c attackers collude; however, this does not represent a significant challenge, since the collusion bound can be set arbitrarily large without significant impact in performance.

Even though our proposal can be used in networks where network coding is not used; it is particularly relevant for network coding, because pollution detection routines based on homomorphic digital signatures such as [6], are computationally intensive. Without proper identification of attackers, node resources can be exhausted by simply flooding the nodes with malformed packets. To circumvent this bottleneck, protocols such as [1], [7] use time as a source of asymmetry; however, this requires loose synchronization among all nodes in the network and packet buffering. These methods can also benefit from our proposal, because information from neighbors and can be used to exclude attackers faster. Other protocols such as [2] are based on a very clever observation; despite packet combinations, the linear span of the original vectors does not change; nevertheless, in order to turn this into a system that can withstand byzantine attackers, either a source of asymmetry or a key distribution scheme is necessary, such as the used in [8].

The rest of the article is structured as follows: in section II, we introduce our problem setting; then in section III we present the primitives needed for our proposal; after this, in section IV we present our proposal to detect an isolate misbehaving nodes; finally in sections V and VI we present our experimental results and conclusions, respectively.

II. Problem Setting and Attack Scenario

There is a wireless network N where $|N| = n$ nodes and a set of incorruptible sources, that wish to send information using random linear network coding. Nodes in the network can be compromised by an attacker, which can control all the

978-1-4673-1881-5/12/$31.00 ©2012 IEEE

resources associated to those nodes. Our goal is to identify the attackers introducing pollution in the network and isolate them; we assume the existence of a pollution detection primitive such as: [1], [2], [6].

We define the security requirement for our MAC based system, in terms of a security game proposed in [9], where no probabilistic polynomial-time adversary has a positive expected payoff when playing the following game: *An attacker can ask to receive the output of the MACs on a sequence of messages* $\{m_1, \ldots, m_g\}$ *of its choice, and then decides to quit or to gamble. If it quits it receives a payment of* 0*. Otherwise, it chooses a message* $m \notin \{m_1, \ldots, m_g\}$ *and tries to guess the value of the MAC on* m*. The adversary receives* $(1-(1/q))$ *if his guess is correct, and pays* $1/q$ *otherwise.*

Intuitively what we want from the security definition, is that our MAC construction cannot be forged by an adversary, even after being exposed to outputs of the MAC function several times. We will assume that nodes can identify their current neighbors before broadcasting; also, there are no more than c compromised nodes during the lifetime of the network.

III. Identifying Malicious Nodes

In this section we will present some existing authentication mechanisms, along with a key assignment needed for our proposal.

A. Digital Signatures

Digital signatures allow any node in possession of a public key P, check that a node in possession of the corresponding private key S, generated a message m; however, the knowledge of P does not allow nodes to produce valid digital signatures. To achieve this property, some kind of asymmetry is needed.

In the RSA signature scheme [10], asymmetry comes from number theory. Compute $n = pq$ (public key), where p, q are prime numbers of a suitable size; then, compute $\phi(n) = (p-1)(q-1)$. Select a number e relatively prime to $\phi(n)$ and find its inverse modulo $\phi(n)$. To sign $0 < m < n$, compute $\sigma = m^d \mod n$; to verify the signature, check if $m = (\sigma^e \mod n)$.

Another source of asymmetry to create digital signatures comes from time. This idea is exploited by the Tesla [11] protocol. The construction assumes nodes are loosely synchronized; using this, nodes can generate signatures that can be checked at a future time:

- Assume there is a one way function h, this is given a value s, computing $h(s)$ is easy; but given $h(s)$ is not possible for an adversary to find s.
- Invoke h, t times using the output of the previous invocation as input for the next one; the value for the first invocation will be s. In mathematical terms, this is: $h_0 = h(s)$, $h_i = h(h_{i-1})$. Publish h_t as the "public key".
- During the first period of time, compute a MAC involving the message m using h_{t-1} as a key to the fast symmetric cryptographic function (e.g. HMAC). Finally, publish the result of the MAC along with h_t. In general, use h_{i-1} as a secret input to a function and publish h_i.

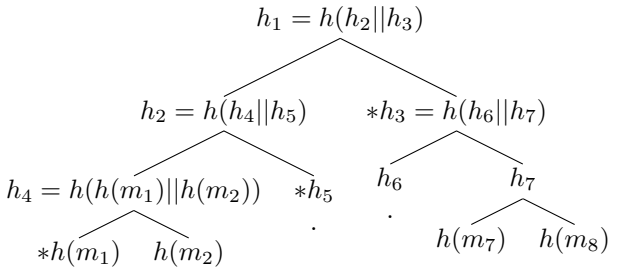

Figure 1. Example of a signature for m_2 using a Merkle tree, appended nodes are marked with $(*)$.

- Nodes cannot check a signature for the current period, but once the period expires, the information is revealed. Security comes from the fact that it is computationally impossible for an adversary to simulate the chain, unless they actually know s.

Since no expensive number theory functions are used, this scheme is very efficient, but incurs in an initial delay. A real time authentication function can be achieved if the source can perform initial packet buffering [12].

In the multicast authentication literature, the Wong and Lam scheme [13], provides fast stream authentication using a binary tree. In this scheme, the root of the tree is delivered reliably. To verify a particular message m_i; $h(m_i)$ and the siblings of its ancestors are appended to the message (h is a cryptographic hash function). Figure 1 contains an example of signature for message m_2, where the nodes of the tree appended to the message have an asterisk $(*)$.

To verify the signature, the verifier simply recomputes the tree using the given information; then if the root matches the digitally signed data, received at a previous stage or from a trustworthy entity, the node can conclude the message is authentic.

B. Blom's Scheme [3]

Blom's scheme allows every user in a network, to share a secret with any other user in an efficient way. Let D be a secret random $c \times c$ symmetric matrix and G be a public $c \times n$ generator matrix of a Maximum Distance Separable (MDS) code; this is a code that meets the Singleton bound: for a $(n, k, d)_q$ then $k = n - d + 1$; Reed-Solomon codes have this property. All elements from both matrices and operations are carried in the field $\mathbb{F}_q$. The set of secrets of the system is given by $A = (DG)^T G$, where $(DG)^T$ denotes the transpose of (DG). The following derivation shows that A is symmetric:

$$A = (DG)^T G = G^T D^T G = G^T (DG) = A^T$$

The shared secret between users i and j is $A_{ij} = A_{ji}$; to compute this value, the i-th user receives the i-th row from the matrix $K_i = (DG)^T$. To get the shared secret A_{ij}; i computes $K_i G_{*j}$, where G_{*j} is the $j-$th column of G; similarly j computes $A_{ji} = K_j G_{*i}$. If $x \leq c$ attackers collude, they get no information about secrets not in the collusion.

C. HMAC [14]

HMAC is a cryptographic MAC, based on a hash function h and a random key k ($||$ denotes the concatenation operator). Given information m to be authenticated

$$\text{HMAC}(k, m) = h((k \oplus \text{opad})||h((k \oplus \text{ipad})||m))$$

where opad=$0x36||\ldots||0x36$ and ipad=$0x5c||\ldots||0x5c$ are fixed values padded to match the key length. If a node wishes to verify that a node in posession of k created a message, it must recompute the function from the received message and then check if the output matches what was received. Proofs of security for HMAC are found in [15].

IV. Proposal

As pointed out in section II, our proposal relies on an existing pollution prevention scheme, which we will denote as a Source Message Authentication Codes (SMAC). MACs introduced by our proposal will be denoted as Relay MACs (RMACs). The key idea of the protocol, is that it guarantees authentication for all messages using a fast function, before the slow SMAC verifying routine is applied.

A. Initialization

The sources initialize every node in the network and themselves, with two sets of secret values S_i and R_i. The purpose of S_i is to give the node the ability to authenticate information sent by the sources; this information will be modified using network coding operations. R_i is given according to Blom's scheme, its purpose is to generate shared keys k_{ij} to authenticate messages among neighbors. Recall from section III-B that these values correspond to the $i-$th row of the matrix $(DG)^T$; in addition, the identifier i is also provided to the node. A digital signature of all the keys in the system is given to each node, along with information to produce a proof that one key was assigned by the source to a particular node. We will now explain how these are generated.

To avoid the use of digital signatures based on number theory at the nodes; we will use the tree construction to sign the whole square matrix of mutual secrets A (section III-A). The idea is to embed A in the tree, as a set of row vectors appended one after another.

Recall from section III-B that the secrets given to node i, are represented by the $i-$th row of matrix A, that will be denoted by A_{i*}. The first step taken by the source is to produce a hash of each row in A independently, by using each coordinate of A_{i*} as a separate message for a Merkle tree, whose root for each row will be denoted by $h_{1,i}$. Figure 2 shows this procedure; the reason we concatenate the actual coordinates of the secret to the invocation of the hash function, is to guarantee the secret was in that given position in matrix A.

Next, create a vector $v = (h_{1,1}, h_{1,2}, \ldots, h_{1,c-1}, h_{1,c})$ where each component is the root of the tree computed from each row. Now we use this vector as an input to another tree whose root will be named σ_A; σ_A is the signature for the whole matrix A. In this computation, the additional string is not needed.

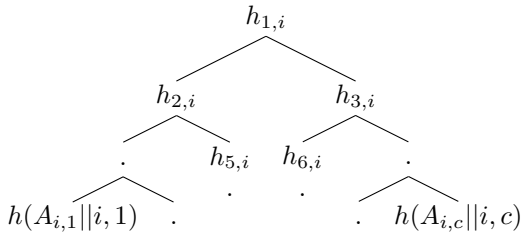

Figure 2. Computing the hash of the i-th row in A (A_{i*}).

id	r	m	SMACS	RMACS

Figure 3. Packet structure

The only remaining thing to give the nodes, is information to prove other nodes, one secret was assigned to them by the sources. To accomplish this, nodes receive, the siblings of their ancestors in the Merkle tree created from v. The reasoning behind this, is that only node i is able to replicate the path for any of its secrets until $h_{1,i}$; then, by using the information from the siblings from his ancestors, any node can verify the result without contacting the source.

B. Transmission

When any node i (including the source) wants to transmit a message, it creates a packet with the following format:

- id: Id of the the last relay that processed the message; for this case i.
- r: Increasing value used to derive a different key for computing RMACs.
- m: Data to be sent.
- SMACs: A suitable pollution prevention scheme.
- RMACs: One RMAC for each one of its neighbors j; first, a master key is derived for each neighbor $\kappa_{ij} = \text{HMAC}(\text{salt}, k_{ij})$, where salt is a random public system parameter; k_{ij} is derived using Blom's scheme. The key used to generate a RMAC for node j is: $k'_{ij} = \text{HMAC}(\kappa_{ij}, r)$. Next, let $m' = (\text{id}, r, m, \text{SMACS})$; the output of one RMAC function is $(\text{id}_j||\text{HMAC}(k'_{ij}, m'||\text{id}_j))$.

C. Relay Processing

When node j receives a packet, it verifies the RMAC intended for it. If the RMAC is not authentic, the packet is discarded immediately; otherwise, the node stores the packet until the SMAC in the packet can be verified (buffering is common in Tesla-based protocols [1]).

If the verification for SMACs is successful, the packet is coded with other packets by generating a linear combination of them; otherwise, the node increases a "bad events counter" for the sender and calls the decision routine (section IV-D); this counter is set to 0 periodically. If the decision routine returns "true", packets from sender i are not relayed anymore.

To inform other nodes of the discovery of a misbehaving node i, entry $A_{ji} = A_{ij}$ is sent to the neighborhood, along with information from the internal part of the Merkle tree only known to node i. We want to point out that the whole internal tree needs to be computed only once, so it can be queried later to accelerate the procedure, the total amount of information stored is twice the number of nodes in the network, times the size of the output of the hash function (e.g. for 10^5 nodes and a 256-bit hash function, memory used is 6.1 MB).

As it was stated in the initialization stage, nodes complement their part of the tree with the message received, if the result of the tree computation equals σ_A, the secret is considered legitimate. Note that the number of hash invocations is $2\log_2(n)$, where n is the number of nodes in the network.

Consider now node l who receives the new secret and who still considers i as honest; then, l uses that key to check the RMAC authenticity of packets coming from i, whose SMACs have not been authenticated yet. Every time a RMAC is verified successfully and the corresponding SMAC is invalid, we increment the counter for "Bad events" proportional to the number of valid RMACs we can check. If the number of bad events from node i exceeds threshold x, l labels i as compromised and reveals $A_{il} = A_{li}$ to his neighbors.

D. Decision Routine

This routine is only called when either: 1. A node presents a valid RMAC for another node; 2. The message authenticated by the SMAC is not authentic. Assuming there are c (or less) compromised nodes, an attacker can cause the first event by either, forging the SMACs for one node and letting an honest node forward the packet (attack I); or forging the RMAC himself (attack II).

In attack I, we assume the attacker can forge a valid SMAC with probability $1/p$; thus, the probability in succeeding in this attack v times is:

$$\left(\frac{1}{p}\right)^v \cdot 1 \tag{1}$$

here, the first term is the probability of an attacker forging the SMAC v times. The second term, is the probability of the second event happening; the value is 1 because an honest node will always forward packets that pass the verification routine for SMACs.

In attack II, the first event occurs with probability $1/|\mathrm{RMAC}|$; hence the probability of the first event happening v times is given by $1/|\mathrm{RMAC}|^v$. If both events happen v times, the probability of labeling an honest node as compromised is given by:

$$\frac{1}{|\mathrm{RMAC}|^v} \cdot \left(1 - \frac{1}{p}\right)^v \tag{2}$$

which is the probability of submitting a valid RMAC, times the probability of the validated message to be wrong by chance. Both events are considered independent because the RMAC and SMAC routines are completely unrelated due to their construction. The system administrator can determine a threshold that is reasonable for false positives depending on the system parameters and equations (1) and (2).

E. Security of the Proposal

Our scheme relies on two protocols, the first of them is the authentication protocol, which relies on secret keys used to generate a random looking one-time key using HMAC.

Using this one-time key, the node invokes HMAC again to produce the definitive output of the RMAC function. For an attacker without the key, a RMAC looks random; thus, the probability of an attacker guessing a valid message-RMAC pair is $(1/|\mathrm{RMAC}|)$. When we apply this to the security game from section II, the result is:

$$\left(1 - \frac{1}{\rho}\right)\left(\frac{1}{\rho}\right) - \left(\frac{1}{\rho}\right)\left(1 - \frac{1}{\rho}\right) = 0$$

where $\rho = |\mathrm{RMAC}|$; satisfying our security requirement.

To prove the protocol construction guarantees authentication in our scenario, consider the following: information needed to create a shared key is known by 3 entities, namely i, j (the two nodes who want to interact in the protocol) and the sources. In our scenario the sources are incorruptible, hence node i knows the MAC was produced by j, because no other node knows the key.

The other situation that needs to be analyzed is when keys are revealed; we consider the two relevant cases:

- A collusion of attackers I reveals the secrets A_{Ij} where j is honest. This is equivalent to the attackers banning themselves from the network.
- A collusion of attackers I reveals the secret A_{Ij} to falsely incriminate j with node l, both j, l are honest. Any RMAC computation by l, using secrets from A_{Ij} as input, is used only after a message has been authenticated using A_{jl}; since A_{jl} is unknown to I, this attack is no better than faking an RMAC for key A_{jl} as long as $|I|$ is within the collusion bound. In practice, special care must be taken to set threshold x to manage this attack properly.

V. Simulation Results

Tests were performed in an Intel Core 2 Quad 2.66 GHz Q6700 and Java JDK 1.6.0_21; all simulation values for times presented, are the average of 10000 runs.

In implementing the authentication protocol, we used HMAC as an RMAC function. Since the protocol involves creating an RMAC for each one of the neighbors, the overhead in this case would be given by the number of neighbors times $|\mathrm{RMAC}|$ bytes in total, plus 2 bytes for the id of the neighbor. Assuming every node has an average of 5 neighbors and $|\mathrm{RMAC}| = 4$, this would imply a total of 30 bytes in overhead. The time to perform the computation of the authentication protocol was 296.3 μs for a message of size 1500 bytes, assuming the keys had already been computed. The actual time for computing a shared secret during our experiments was

Table I
EXPERIMENTS IN C

Algorithm	Sign(μs)	Verify(μs)	Output(bits)	Security
RSA	3371	86	1024	2^{80}
RSACRT	1019	86	1024	2^{80}
RSACRTLE	1013	30	1024	2^{80}
DSA	450	790	320	2^{80}
Tesla	2	2	320	2^{80}
Proposal 32 bit	10	10	240	2^{32}
Proposal 80 bit	10	10	400	2^{80}

around 1 ms, for a collusion bound of 1000 attackers, where arithmetic was performed modulo a prime close to 2^{128}.

To test how our method introduces additional delays, we implemented two fast, SMAC mechanisms based on the product of random vectors in a finite field, one scheme was [1] and the other was [8]. The first of them computes the product of a random vector with each vector to be authenticated, then it delays the disclosure of the key used to generate the random vectors using the Tesla protocol; the implementation of the protocol with an output of 1 byte, took approximately 20 μs including the generation for the Tesla signature and the random vector using a SHA-1 based random number generator from the Java Crypto API. The second one is similar but encrypts the output of the random vector to generate a MAC, our times for a packet with 5 MACs were about 300 μs.

We evaluated authentication times using C++; the environment was g++ 4.6, GNU Multi-Precision library $5.0.1-11$ for big numbers, and the SHA-1 implementation from Crypto++ $5.6.1-5$. The algorithms we measured were RSA, DSA, Tesla based authentication and our MAC-Based proposal; results are summarized in Table I. Results show that number theoretic digital signatures are slower than their symmetric version counterparts, as it was anticipated; however, in their favor they provide verification for an unbounded number of nodes.

The last two symmetric schemes are significantly lighter in terms of computation and overhead. The Tesla solution as implemented by ourselves, appends the previous hash in every packet. On the other hand, our construction scales linearly in time with the number of neighbors, for this case 5 neighbors were used. Even though our construction is tailored to be used in the local vicinity of a given node, it improves the Tesla based construction since packets can be forwarded immediately with the same security guarantees. Tag pollution is not an issue since the tags are only meant for one hop.

The amount of storage needed by our authentication protocol is as follows: 1. c numbers as part of the matrix to generate the secret keys; 2. $160 \cdot \log_2 c$ bits hashes for the path of the tree needed. For our experiments with 2^{16} nodes, the total amount of information stored by the node was 16320 bytes, assuming a collusion bound of 1000 nodes. This contrasts with only one output of a hash function in [1] (160) bits; despite this our proposal is reasonable within the technical specifications of contemporary devices. Our system has similar memory requirements when compared to MAC-based systems used for pollution detection.

VI. CONCLUSIONS

We presented a computationally light mechanism to isolate malicious nodes in a network, that works in the presence of a great number of attackers; unlike other proposals in the literature, we do not rely on the existence of a central authority to perform this duty, or any particular topology; this holds as long as the underlying pollution detection routine is secure.

ACKNOWLEDGMENTS

This work was supported in part by a Grant in Aid for the Global Center of Excellence Program for "Center for Education and Research of Symbiotic, Safe and Secure System Design" from the Ministry of Education, Culture, Sport, and Technology in Japan.

REFERENCES

[1] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical defenses against pollution attacks in wireless network coding," *ACM Trans. Inf. Syst. Secur.*, vol. 14, pp. 7:1–7:31, June 2011.

[2] P. Zhang, Y. Jiang, C. Lin, H. Yao, A. Wasef, and X. Shen, "Padding for orthogonality: Efficient subspace authentication for network coding," in *INFOCOM, 2011 Proceedings IEEE*, april 2011, pp. 1026 –1034.

[3] R. Blom, "Non-public key distribution." in *Advances in Cryptology: Proceedings of CRYPTO '82*, 1982, pp. 231–236.

[4] A. Le and A. Markopoulou, "Cooperative defense against pollution attacks in network coding using spacemac," *CoRR*, vol. abs/1102.3504, 2011.

[5] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "Mis: Malicious nodes identification scheme in network-coding-based peer-to-peer streaming," in *INFOCOM, 2010 Proceedings IEEE*, 2010, pp. 1–5.

[6] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a linear subspace: Signature schemes for network coding," in *Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography: PKC '09*. Springer-Verlag, 2009, pp. 68–87.

[7] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen, "Ripple authentication for network coding," in *INFOCOM, 2010 Proceedings IEEE*, march 2010, pp. 1 –9.

[8] S. Agrawal and D. Boneh, "Homomorphic macs: Mac-based integrity for network coding," in *Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, vol. 5536, pp. 292–305.

[9] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast security: a taxonomy and some efficient constructions," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, mar 1999, pp. 708 –716 vol.2.

[10] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 26, pp. 96–99, January 1983.

[11] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "The tesla broadcast authentication protocol," *RSA CryptoBytes*, pp. 2–13, 2002.

[12] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Efficient and secure source authentication for multicast," in *In Network and Distributed System Security Symposium, NDSS '01*, 2001, pp. 35–46.

[13] C. K. Wong and S. Lam, "Digital signatures for flows and multicasts," in *Network Protocols, 1998. Proceedings. Sixth International Conference on*, oct 1998, pp. 198 –209.

[14] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '96. London, UK: Springer-Verlag, 1996, pp. 1–15.

[15] *New proofs for NMAC and HMAC: Security without collision-resistance*, ser. Lecture Notes in Computer Science, vol. 4117. Springer, 2006.