PEAC: a Probabilistic, Efficient, and resilient Authentication protocol for broadcast Communications

Andrea Cirulli
Valueteam S.p.a.
Via della Grande Muraglia 284
Roma, Italia
andrea.cirulli@valueteam.com

Roberto Di Pietro*
UNESCO Chair in Data Privacy
Universitat Rovira i Virgili
roberto.dipietro@urv.cat

ABSTRACT
One of the main challenges of securing broadcast communications is source authentication: to allow each receiver to verify the origin of the data. An ideal broadcast authentication protocol should be efficient for the sender and the receiver, have a small communication overhead, allow the receiver to authenticate each individual packet as soon as it is received (i.e. no buffering on the receivers), provide perfect robustness to packet loss, and scale to a large number of receivers.

In this paper we introduce PEAC, a probabilistic, efficient and resilient authentication protocol for broadcast communications. This new construction achieves all the above properties, with the tradeoff that it requires just loose time synchronization between the sender and the receivers. Due to its low communication overhead and minimal synchronization requirements, the scheme is particularly suitable for low-end resource constrained devices as well as applications that require to process the received messages in real time or quasi-real time fashion. For instance, a packet can be authenticated computing 12 hash only on both the sender and the receivers, while the packet forging probability is kept below $2^{-80}$. Finally, note that PEAC is completely customizable, allowing to trade-off security with a (small) overhead increase on the sender only, while not affecting the (small) overhead experienced by receivers.

Categories and Subject Descriptors
C.2.0 [Computer-Communication Networks - General]: Security and protection; K.6.5 [Security and Protection]: Authentication

*The author is solely responsible for the views expressed in this paper, which do not necessarily reflect the position of UNESCO nor commit that organization. He is also with Università di Roma Tre, Dipartimento di Matematica.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2008 ACM 978-1-60558-241-2 ...$5.00.

General Terms
Design, Security, Performance

Keywords
broadcast authentication, probabilistic algorithm

1. INTRODUCTION
Single-source broadcast is a compelling mechanism of communication since it enables the sender to efficiently disseminate data to a large audience. Some interesting broadcast distribution networks are IP multicast, satellite communication, or wireless communication. Moreover, bogus packet injection is easy in many broadcast networks; hence, communication protocols have to assure that the packets are really originated by the claimed source. A broadcast authentication protocol enables the receivers to check the validity and origin of the received packets. Applying the standard unicast authentication mechanism (i.e. appending a HMAC to each message, computed by a shared secret key) to the broadcast environment does not assure authentication. In broadcast setting some sort of asymmetry is needed between the sender and the receivers, such that only the sender is able to generate the authentication token for a message, while the receivers can only validate messages. However, applying asymmetric cryptography to provide authentication in broadcast communication is not always suitable, due to the expensive computational overhead that asymmetric primitives involve, such as digital signatures. One approach to deal with the computational overhead of a digital signature is to amortize it. Signature amortization [3, 8, 14] addresses this issue by generating a single digital signature for a block of packets. The main drawback of signature amortization protocols is that they are not tolerant to packet loss, which is an important property in an unreliable communication media, such as the broadcast one. To tackle this drawback, signature amortization protocols are combined with erasure codes [6, 7] that partition information into many segments and add some redundancy to them; thus enabling the receivers to rebuild original data even in presence of packet loss. However, erasure codes suffers from pollution attacks [4]. Pollution attacks occur when an adversary injects malicious packets into the communication stream, making the receivers unable to decode original data from this polluted stream. The first scheme proposed to deal with pollution attacks [4] is based on distillation codes. In distillation codes, the sender appends to each packet a witness. Upon receiving the entire stream of packets, the receiver separates packets...
into sets by witness. However, a receiver does not know in advance which set is composed of valid packets only, and it has to decode and verify the signature of each set. Furthermore, this approach involves buffering on receivers side, and this is not suitable for real-time applications or when receivers are low-end, resource constrained devices.

The contribution of this paper is to present PEAC: a probabilistic, efficient, and resilient lightweight broadcast authentication protocol. Our protocol accomplishes all the requirements needed for broadcast authentication; PEAC is: efficient as for computations and memory required, has a small communication overhead, allows receivers to authenticate each individual packet as soon as it is received (i.e. no buffering is required), provides perfect robustness to packet loss, and scales to a large number of receivers. Though it needs just loose time synchronization between the sender and the receivers, it is suitable for real-time applications, because it does not require buffering neither on the sender side nor on the receivers side. Further, the trade-off between security achieved and overhead incurred is completely customizable.

The rest of the paper is organized as follows: in Section 2 we introduce related work; in Section 3 we detail some assumptions and backgrounds that underlie our scheme. PEAC, our proposed scheme, is introduced in Section 4. An analysis of PEAC, in terms of provided security and introduced overhead is in Section 5. Finally, in Section 6 we draw the conclusions.

2. RELATED WORK

One possible approach to assure authentication in broadcast communication assumes that sender and receivers are loosely synchronized. This approach limits the lifetime of keys used to authenticate packets, to prevent that an adversary could use disclosed keys to forge an authenticated packet. TESLA [12] is an instance of such schemes: TESLA uses the HMAC function to authenticate messages, and one-way chains to produce HMAC keys. A secret HMAC key used to authenticate a message in a time interval is kept secret by the sender, to prevent an adversary to use this key to forge an authenticated message. Upon key expiration, the sender discloses the key used to authenticate messages sent within this elapsed time. Receivers use the disclosed key to verify the authenticity of previously received messages. To prevent an adversary to use a disclosed key to forge a packet that could pass authentication check, sender and receivers must be synchronized. The bandwidth overhead induced by TESLA is one HMAC output size, plus the disclosure of one key per period of time. The main drawback of TESLA is that receivers must buffer received packets until the corresponding key is revealed. This make TESLA not suitable for real-time broadcast applications and resource constrained devices.

Another approach to provide broadcast authentication is signature amortization [3, 14, 17, 18]. Signature amortization protocols divide the information stream into blocks of sequential packets which are all authenticated with a single signature. This is a compelling approach to provide broadcast authentication, because it amortizes the computation and bandwidth overhead of a digital signature over many packets.

One issue of signature amortization protocols is tolerance to packet loss. To authenticate packets in a block receivers need to receive the digital signature of the packets. Appending the signature in every packet implies waste of bandwidth. Conversely, including few bytes of the signature in each packet is efficient but not tolerant to packet loss. Three seminal solutions were proposed to tolerate data loss in signature amortization: hash graphs [14, 8, 3]; the Wong-Lam scheme [18], and schemes [9, 10] using erasure codes [16, 6, 7]. Several researcher support the use of the erasure codes for signature amortization. Erasure codes are a technique that enable receivers to decode messages from a sufficiently large quorum of encoding symbols. One mechanism to tolerate packet loss is the Information Dispersal Algorithm (IDA) [16]. IDA consists of an encoder and a decoder. IDA encoding complexity is \(O(d)\), where \(d\) is the original data size, and decoding complexity is \(O(d(\log q + r))\), where \(n = q + r\) is the size of erasure codes encoder output, using the algorithm proposed in [15].

Other erasure codes are Tornado [7] and LT [6]. They require linear encoding and decoding, but they are less space efficient than IDA.

Signature amortization schemes combined with erasure codes are exposed to pollution attack. In pollution attack an adversary can inject invalid symbols into the communication stream. If an erasure code uses just an invalid symbol to reconstruct the data, it will yield invalid data. The communication model that describes this issue is the polluted erasure channel. In this model packets can be lost, and an adversary can inject invalid symbols and delay packets. Karlof et al. [4] propose distillation codes to deal with pollution attack. In this approach, the sender builds a Merkle Tree using the hash value of the broadcast packets. Then, for each packet, the sender constructs and appends a witness, the verification nodes of a Merkle Tree. When a receiver gets the packets, it separates them in many sets according to each packet’s witness. Distillation codes ensures that exists a set containing only valid packets, allowing the receiver to successfully recover the original data. This approach has an overhead of one signature per block, plus \(\log n\) hash per packet, where \(n\) is the length of the block. The protocol in [1] proposes an optimization of Merkle tree distillation codes, named Pruned Merkle Tree (PMT), and an algorithm (CECInA) for encoding/decoding that mitigates pollution attack. CECInA, in combination with PMT, achieves better performance, in terms of bandwidth occupation and computation power than the scheme proposed in [4].

In [19] a new approach is proposed to resist to pollution attacks, named PARM. The main idea behind PARM is to append an evidence to each packet to prove its authenticity. PARM scheme consists of four phases: initialization, evidence generation, evidence validation, keys renewal. These phases are sub-components of the PEAC protocol as well, and they are described in Section 4.

During the initialization phase, the sender generates a temporal key pair consisting of a temporal public key (TPK) and a set of temporal secret key chains (TSK). As in the asymmetric encryption: the sender uses the TSK to build evidence for a message, while the receivers validate the evidence through the TPK. In order to create a temporal key pair the sender first generates \(k\) \(n\)-bit random numbers \(\{R_0, \ldots, R_{k-1}\}\) denoting this set as \(\text{TSPK}_0\). Once generated this set of random numbers it recursively applies, for \(L\) times, the hash function \(h\) to each random number,
building \( k \) hash chains of length \( L \). The TSK consists of \( L \) sets \( TSK = \{ TSK_0, \ldots, TSK_{L-1} \} \), each set is formed by all the elements hashed the same number of times starting from the \( TSK_0 \). For example, the \( TSK_1 \) consists of \( \{ h(R_1), h(R_2), h(R_3), \ldots, h(R_{L-1}) \} \) the same for the other TSKs chains. The TPK consists of the \( L \)-th hash of each random number generated by the sender, that is: \( TPK = \{ h^L(R_0), h^L(R_1), \ldots, h^L(R_{L-1}) \} \). Let \( TSK_{y}^u \) be the \( y \)-th TSK chain and \( y \) the \( y \)-th element of the considered TSK chain, and let \( TPK_{w} \) be the \( w \)-th element of the TPK. A message is signed in the generation phase. This phase is prior to communication and consists in appending \( p \) elements of TSKs chains bound by the message to sign. To generate the evidence \( E \) for a message \( M \), the sender uses the hash function \( h \) to perform \( h(M) \). Next, the sender splits \( h(M) \) into \( p \) segments \( S = \{ S_0, \ldots, S_{p-1} \} \) of at most \( b \) bits, where \( b = \lceil \log L \rceil \). By interpreting each segment as an integer it can index a specific TSK chain for each segment. For each index \( S_i \), the sender determines the TSK chain on which choosing an element, that is \( TSK_{S_i} \). Once chose the TSK chain it has to select the first unused value of that TSK chain, that is by selecting upon the usage of \( TSK_{S_i} \). Hence, \( TSK_{S_i}^{u+1} \) is selected where \( u \) denotes the usage of \( TSK_{S_i} \). The sender has to keep trace of the usage of each TSK chain hence it updates \( u_{S_i} \). For example, the sender will select \( TSK_1 \) given \( S_0 = 1 \) and given that three elements of \( TSK_1 \) are already revealed, that is \( u_1 = 3 \). It will do the same for all the indexes in \( S \). In this way the sender builds the set \( E = \{ e_0, \ldots, e_{p-1} \} \), where \( e_i = TSK_{S_i}^{u+1} \). Then, the sender sends a packet \( P = (M, E) \).

Upon receiving the authenticated packet, the receivers can authenticate the packet by validating the evidence included in it. To determine the validity of the packet the receivers must use two sets of data: the TPK sent by the sender and the evidence set that involves the hash values contained within the evidence. When the packet is received the receivers separate the message \( M \) from its evidence \( E \). To validate \( E \) the receivers just need a subset of the TPK they have. As a matter of fact, the receivers have to select by the message \( M \) which elements of the TPK are needed to validate \( E \). To build this set, the receivers first compute \( h(M) \) and then split the outcomes into \( p \) segments \( S = \{ S_0, \ldots, S_{p-1} \} \) of \( b \)-bits as the sender, and treat the resultant integers as indexes to the TSK chain. Each index \( S_i \) of the TSK chain, along with its usage \( u_{S_i} \) (i.e. the number of elements of \( TSK_{S_i} \) already revealed by the sender), determines the number of times to hash the corresponding element \( e_i \) of the evidence \( E \) and the TPK element that has to be obtained by hashing \( e_i \). Given an index \( S_i \) and its usage \( u_{S_i} \), the receivers should perform \( L - S_i \) hashes on the corresponding element of the evidence \( e_i \) of \( E \) to obtain \( TPK_{u_{S_i}} \). Thus they build a verification subset \( V_S \) of the TPK by selecting for each index \( S_i \) the corresponding TPK element \( TPK_{u_{S_i}} \), then \( V_S = \{ TPK_{u_{S_0}}, \ldots, TPK_{u_{S_{p-1}}} \} \). The receivers have to check if \( E \) is valid through \( V_S \). Finally, the receivers accept the packet if every element \( e_i \) of \( E \) hashed \( L - S_i \) times is equal to \( TPK_{u_{S_i}} \), even if one check fails the packet is rejected.

Since PARM does not involve reuse of any hash value, the authors define a threshold value \( T \) in the key renewal phase. \( U_{TSK_0} \) represents the number of used elements in \( TSK_0 \) (the first TSK of the TSK chain). When \( U_{TSK_0} \) exceeds the threshold \( T \), new elements are required. First, the sender generates \( U_{TSK_0} \) new random numbers for the used indexes of \( TSK_0 \). Using these random numbers, the sender creates the partial TSK and the partial TPK with the one-way hash function \( h \) by following the temporal key generation procedure of the initialization phase. Finally, it sends the new partial TPK to the receivers.

3. BACKGROUND AND ASSUMPTIONS

PEAC is built over one-way chains, in conjunction with loose time synchronization mechanisms [13, 12]. In this subsection we first briefly review one-way chains, later the threat model assumed throughout the rest of the paper, and finally we list the features a broadcast authentication protocol should enjoy.

3.1 One-Way chains

A one-way chain can be used to commit a sequence of pseudo-random values. To achieve this purpose, we repeatedly use a one-way hash function to produce a one-way chain; this cryptographic mechanism, introduced by Lamport [5], is widely used to design security protocols. To produce a chain of length \( L \) we first choose a random value that will be the last element of the chain \( s_l \). To generate the remaining elements of the chain, we repeatedly apply a one-way function \( h \) to the first element \( s_l \). Eventually, \( s_0 \) is the commitment element for the entire chain, and we can check the validity of any element of the chain through the commitment \( s_0 \). In particular, to check the validity of the element \( s_i \), where \( i \) denotes the \( i \)-th element of the chain, we verify that \( h^{i}(s_i) = s_0 \). In general, given a \( s_i \), it is possible to authenticate any \( s_j \) with \( i < j \). To check whether \( s_j \) is an element of the chain or not, we just verify that: \( h^{j-i}(s_i) = s_j \). For this reason, elements of the hash chain are revealed in reverse order of generation, that is: \( s_0, s_1, \ldots, s_l \).

3.2 Threat Model

In this paper we assume that the adversary is compliant with the model proposed by Dolev-Yao [2]. Therefore, the adversary is able to eavesdrop, to inject, to modify, and to drop transmitted packets. In particular, the adversary can execute the following actions: (1) intercept and learn any message; (2) drop chosen messages; (3) introduce forged messages into the communication channel. The devised attack scenario is characterized as follows:

- the adversary generates a message \( M' \), and it wants to forge authentication for \( M' \);
- the adversary intercepts any authenticated message sent by the sender;
- the adversary can learn the authentication elements revealed in each authenticated message.
- if the adversary is able to collect all the required authentication elements needed to authenticate \( M' \), the authentication check for the message \( M' \) will succeed, and message authentication is violated.

For the above threat model, we will show that PEAC provides higher probabilistic resilience when compared to a state of the art solution (PARM [19]).
3.3 Broadcast authentication features

An ideal broadcast authentication protocol should enjoy the following features:

- efficient generation and verification of authentication elements;
- low bandwidth overhead;
- no packets buffering on both the sender and on the receivers;
- packet loss tolerance;
- scalability: authentication elements size is independent from the number of receivers.

We will show in the following that the PEAC protocol matches all the above features, provided that the sender and the receivers are just loosely time synchronized.

4. PEAC

PEAC leverages one-way hash chain and loose time synchronization to achieve packet authentication. In this section we will detail PEAC. In particular, it consists of four phases: initialization, authentication elements generation, packets validation and temporal keys renewal.

4.1 Initialization

This phase defines how to produce a temporal secret/public keys pair. This pair consists of a temporal secret key chain (TSK) and a temporal public key (TPK). Both the TSK and the TPK are built using a one-way hash function. The TSK-chain is used by the sender to produce the authentication elements for a packet, while the TPK is used by the receivers to check packet authenticity.

The TSK chain and the TPK must be generated by the sender prior to the communication. The generation process is the following: the sender produces \( n \) random numbers of \( k \)-bits \( (R_0, R_1, \ldots, R_{n-1}) \); this set of numbers is referred to as TSK_0 of the TSK-chain. Afterwards, the sender applies the one-way hash function \( h \) to each member of the previous TSK to obtain the next TSK. In particular, to obtain a TSK-chain of length \( L \), the sender generates TSK_1 by applying \( h \) to each element of TSK_0 i.e. TSK_1 = \( \{h(R_0), h(R_1), \ldots, h(R_{n-1})\} \). The TPK is generated by applying the one-way hash function \( h \) to each element of the last TSK of the TSK-chain (e.g. with a TSK-chain of length \( L \), the TPK is produced applying \( h \) to each element of TSK_{L-1}). Next to the generation of both the TSK-chain and the TPK, the sender splits up time into time intervals of uniform duration. The sender sequentially assigns each TSK of the TSK-chain to a time interval, that is one TSK per time interval. The TSK-chain is used in reverse order of generation. Hence, TSK_{L-1} is assigned to the time interval \( t_0 \). In general, the TSK assigned to a time interval \( t_i \) is TSK_{(L-1)-j}. Finally, the sender must transmit the temporal public key TPK to all the receivers. Since the TPK assures the authenticity of a packet, the sender digitally signs the TPK, and broadcasts it through a reliable channel. Each receiver, after having checked the validity of TPK, stores it and subsequently will use it to verify the authenticity of the packet.

Figure 1 shows the procedure for TSK and TPK generation. The first TSK of the TSK-chain (TSK_0) is generated by picking \( n \) random numbers of \( k \)-bits each. The arrows represent the application of the one-way function \( h \). Hence, \( h(R_0) \) represents the application of \( h \) to \( R_0 \), and \( h'(R_0) \) is \( h(h'(R_0)) \). The collection of elements of the same row constitutes a TSK element of the TSK-chain, that is TSK_0 = \( \{R_0, \ldots, R_{n-1}\} \) and the last element of the TSK-chain is TSK_{L-1} = \( \{h^{L-1}(R_0), \ldots, h^{L-1}(R_{n-1})\} \).

The elements of the last row constitute the public key TPK.

4.2 Authentication element generation

To authenticate a message \( X \) the sender must generate a set of authentication elements for \( X \). The sender uses the TSK to generate such set of authentication elements, that enable receivers to authenticate the message. The set of authentication elements consists of \( p \) elements of a TSK_{j}. In order to avoid from replay attack, the sender signs not just the message \( X \) but the message \( X \) prefixed by a time stamp. Let \( T \) be the time stamp, that is a sequence of characters, denoting the date and/or time at which a certain message is signed. The sender prefix the time stamp \( T \) to the message \( X \), let us denote this packet as \( M = T \| X \). In particular, to generate a set of authentication elements \( E_M = (t_0, \ldots, t_{p-1}) \) for the message \( M \) to be sent in time interval \( t_j \), the generation phase is the following:

- the sender determines the TSK of the TSK-chain assigned to the time interval \( t_j \) that is: TSK_{(L-1)-j}. Thus the sender uses only elements selected from TSK_{(L-1)-j};
- the sender stores a usage table employed to keep track of which elements of the active TSK were already revealed to support the authentication of previous messages (within the current time slot);
- the sender produces the set \( S = (t_0, \ldots, t_{p-1}) \) and the set \( C_M = (c_0, \ldots, c_{p-1}) \) in this way: the sender first sets a counter \( c = 0 \) and computes \( t_0 = h(M\|c) \mod n \), where \( n \) is the length of the TSK of the TSK-chain. Then the sender increments the counter \( c = c + 1 \) and computes \( t_1 = h(M\|c) \mod n \); this procedure is iterated to obtain all the \( p \) elements. Note
that collisions can happen, where a collision occurs when \( i_j = i_k \) for some \( j > k \geq 0 \). In this case, the sender just does not accept the value \( i_j \), but simply increments the counter and iterates the procedure until a value \( i_j \) not colliding with any of the generated values so far, for the active chain–, is obtained. The ordered set \( C_M = (c_0, \ldots, c_{p-1}) \) is given by the values of the counters that contributed to compute an accepted value \( i_j \). An analysis of the overhead introduced to avoid collision is given in Section 5.3:

- the set \( S \) determines the elements of the active TSKs that will be used to construct the set of authentication elements \( E_M \), where \( E_M = (e_0, \ldots, e_{p-1}) \) is generated as follows: for each index \( i \in S \), the sender selects the \( i \)-th element of the chain \( TSK_{(L-1)} \) active for that time interval \( t_i \) —for instance, \( e_0 \) can be the \( i_0 \)-th element of \( TSK_{(L-1)} \). In this way, the sender produces the \( p \) elements that constitute the authentication elements for the message \( M \);

- finally, the sender appends to the message \( M \) the set \( E_M \) and the set \( C_M \) and then broadcasts the packet.

Figure 2 depicts the authentication element generation process. For instance, to produce the set of authentication elements \( E_M = (e_0, \ldots, e_{p-1}) \) for the time interval \( t_i \) to authenticate a message \( M \), the sender chooses \( p \) elements of the active TSK, (note that in time interval \( t_i \) the active TSK is \( TSK_{(L-2)} \)). The message \( M \) drives the selection process for the authentication elements. After choosing the \( p \) elements of the TSKs, the sender appends the set of authentication elements \( E_M \) and the set \( C_M \) to the message \( M \) (obtaining \( \langle M, E_M, C_M \rangle \)), where \( E_M \) and \( C_M \) are ordered sets, and broadcasts these data. Algorithm 1 fully details the procedure to generate authenticated messages.

**Algorithm 1**: PEAC: authentication elements generation procedure.

```plaintext
Input : A message \( M \) to authenticate, the time interval \( j \), the security parameters \( n, p \), the temporal private key chain \( TSK \), the number of messages (\( r \)) revealed during the time interval \( j \), and the vector \( H[rp] \) that contains all the authentication elements revealed in the time interval \( j \).
Output: A packet and its authentication elements.

begin
  Let \( t_j \) be the current time interval;
  \( c \leftarrow 0 \);
  \( count \leftarrow 0 \);
  Let \( S \) be a vector of \( p \) elements;
  Let \( C_M \) be a vector of \( p \) elements;
  repeat
    \( i \leftarrow h(M[k]) \mod n \);
    if \( (i = S[x] \text{ for some } x < count) \) and \( (i = H[m]) \text{ for some } m < rp \) then
      \( c \leftarrow c + 1 \);
    else
      \( S[count] \leftarrow i; \quad C_M[count] \leftarrow c; \quad c \leftarrow c + 1; \quad count \leftarrow count + 1 \);
  until \( count < p \);
  Let \( E_M \) be a vector of \( p \) elements; for \( x \leftarrow 0 \) to \( p - 1 \) do
  \( \sigma \leftarrow \epsilon_x \);
  return \( \sigma = \langle M | E_M | C_M \rangle > \);
end
```

The receivers check whether the time stamp is new or not. If the time stamp is obsolete, that is the time stamp was included in a previous authenticated message, it means that this message is just replayed and then the receivers do not take into account that message.

The procedure to verify an authenticated packet \( \sigma \) in a time interval \( t_j \) is reported in the following:

- the receiver knows on which TSK the sender constructed the authentication elements set. This is possible due to the loosely time-synchronization with the sender. For example, in a time interval \( j \) the authentication elements are built upon \( TSK_{(L-1)} \);

- the receiver computes the elements of the set \( S = (i_0, \ldots, i_{p-1}) \) as: \( i_b = h(M[k]) \mod n \) for \( b = 0, \ldots, p-1 \). That is, the receiver leverages both the message and the values used to avoid collision sent by the center. Once obtained the set \( S \), the receiver can proceed checking the validity of the message;

- for each index \( i_b \in S \), the receiver verifies whether \( h^b(\epsilon_{i_b}) \) is equal to \( TPK_{i_b} \) (\( j \) is the time interval). If just one of these checks fails, the packet is rejected, otherwise it is authenticated.

Algorithm 2 fully details the packets validation procedure.

### 4.3 Packets validation

Upon receiving a packet \( \sigma = \langle M, E_M, C_M \rangle \), the receivers can verify the authenticity of the message \( M \) by checking the validity of the authentication elements using the TPK. Since the message \( M \) is characterized by the time stamp \( T \) and the original message \( X \), once authenticated the message

![Figure 2: PEAC authentication elements generation.](image)

### 4.4 Temporal keys renewal

The sender uses the TSK-chain to authenticate messages. Hence, after \( L \) time intervals, where \( L \) is the number of TSK that constitutes the TSK-chain, the key chain is exhausted and the sender must switch to a new TSK-chain and TPK. One approach is to repeat the initialization phase previously described (see Section 4.1). Another approach to solve this problem follows: before the TSK-chain expires,
and once the sender has computed the new temporal keys pair, the sender processes the new TPK as a normal packet to be authenticated via the PEAC mechanism. Upon correctly reception and authentication of the new TPK by the receivers, the sender and the receivers may continue their secure broadcast communication, validating and verifying packets through the new TSK-chain and TPK chain. Finally, note that the sender could also set the length of the TSK in such a way that, from a practical point of view, there could be no need to renew the TPK.

5. ANALYSIS AND DISCUSSION

In this section we provide the security analysis of PEAC. To show its superiority with respect to PARM, we show an attack on PARM [19] and how it is prevented in PEAC. In particular, PEAC leverages the advantages provided by PARM, but provides stronger security, and does not need to buffer received messages, unlike TESLA. These features are trade-off by a slight additional computational overhead with respect to TESLA and the need for loose time synchronization when compared with PARM. We further provide an analysis of the overhead incurred by PEAC.

5.1 Security analysis

To authenticate a message, the sender needs to disclose $p$ authentication elements. In particular, we assume that in a time interval the sender sends up to $r$ authenticated messages. Note that the $m = rp$ disclosed elements of the active TSK could be used by the adversary to forge authentication for a message $M'$. In the following we analyze the probability that such an event could happen, given that $m$ authentication elements have been disclosed by the sender.

Let $n$ be the number of elements in a TSK length (we refer to $n$ as the length of the TSK). We can model the attack under this scenario:

- assume to have a set of $n$ elements that represent all the authenticated messages that could be used in a time interval, let us denote this set as $N = \{N_1, \ldots, N_n\}$, where each elements is an authentication element of the active TSK;
- let $m = rp$ be the number of authenticated messages sent, and let $R = \{R_1, \ldots, R_r\}$ be the set of authenticated elements revealed by the sender, that is $R \subseteq N$.

Given a message $M'$ the adversary wants to forge authentication for this $M'$. This drives the adversary to select $p$ elements of $N$, that is the authentication elements it needs to forge authentication for $M'$. Denote these $p$ elements as: $B = \{B_1, \ldots, B_p\}$. The adversary can authenticate message $M'$ if, after revealing $m = rp$ authenticated messages, all the elements in $B$ are revealed.

Theorem 1 provides the probability of a successful attack on PEAC in relation to the number of revealed authenticated messages per period of time.

**Theorem 1.** Given $m = rp$ authentication elements of the active TSKs of length $n$ in a time interval $t$, the probability $Pr(A)$ that an adversary collecting these authentication elements can forge authentication for a message $M'$ is:

$$\Pr(A) = \frac{\binom{r}{p}}{\binom{n}{p}}$$

**Proof.** Let $N = \{A_1, \ldots, A_n\}$ be the set of authentication elements of the active TSK chain used to authenticate messages in a time interval $t$. The set $N$ is known only by the sender. Each time the sender multicasts an authenticated message it reveals $p$ different elements of $N$. Hence, after revealing $r$ authenticated messages, the sender disclosed $rp$ elements of $N$, where $rp \leq |N|$. Let $R = \{R_1, \ldots, R_r\}$ be the set of disclosed elements of $N$, with $R \subseteq N$. An adversary intercepting all the $r$ authenticated messages can collect $rp$ elements of $N$. Let us assume that the adversary wants to forge authentication for a message $M'$. In order to do that, the adversary performs the generation phase of PEAC on $M'$, without knowing any elements of the active TSK chain, that is $N$. However, it knows which are the $p$ elements of $N$ it needs to forge authentication for $M'$; let us denote this set with $B = \{B_1, \ldots, B_p\}$, where $B \subseteq N$. If after revealing $r$ authenticated messages the adversary has collected all the elements needed to forge authentication for $M'$, PEAC is violated. Since the authenticated elements of $N$ revealed after $r$ messages is the set $R$, and the set of authenticated elements needed by the adversary is $B$, the adversary can forge a signature for $M'$ if $B \subseteq R$. Let $A$ be the event that $B \subseteq R$. Let $A$ be the event that $B \subseteq R$, where $N$ is a set of $n$ elements, $B$ is a set of $p$ elements and $R$ is a set of $rp$ elements, where $B \subseteq N$ and $R \subseteq N$. Let $A$ be the event that PEAC is violated. From the above discussion, it follows that:

$$\Pr(A) = \frac{\binom{r}{p}}{\binom{n}{p}}$$
bound for probability less than $2^{-64}$ and $2^{-80}$, for parameters $n = 2048$ and $p = 10, 12$. In figures 4(a), and 4(b), the x-axis represents the number of revealed messages per time interval, while the y-axis maps the probability of forging an authenticated packet, with the security parameter $n = 2048$. These plots focus on forging probabilities such as $2^{-64}$ and $2^{-80}$ to show that PEAC is completely customizable and that it can reach high level of security. For example it can be noticed in Figure 4(a) that, revealing less than four messages per time interval when $p = 10, 12$, the adversary has a probability less than $2^{-64}$ to forge an authentication packet; this probability drops to less than $2^{-80}$ (Figure 4(b)) when less than two messages per interval time are sent (with $p = 10, 12$). Furthermore, these plots confirm the intuition that security depends on both the number of authenticated messages sent by the sender and the number of authentication elements used to authenticate a message ($p$). In particular, the parameter $p$ shows an interesting effect on security: if the number of authenticated messages sent by the sender per time interval is low, $p = 10, 12$ provides better security than $p = 6, 8$. However, as the number of authenticated messages sent by the sender increase, a better choice would be to set $p$ to values lower than $p = 10, 12$. The influence of $p$ on the security can be explained noticing that the total number of authentication elements revealed by the sender in a time interval, is $m = rp$. Hence, this suggests that to improve security, one direction could be to reduce the number of authenticated messages sent by the sender per time interval. Note that this solution does not introduces drawbacks for PEAC, since this solution implies just to use longer hash chain on the server side, while the receiver side is not affected at all.

For what concerning the replay attack, the chance of doing replay attack is thwarted by the use of time stamps. The sender numbers sequentially all the messages within an interval and prefix the message number to the original message before triggering the message authentication procedure. Hence, security is completely customizable by the sender; further, tuning security does not affect the overhead incurred by the receivers.

5.2 Attack on PARM

In this section we provide an analytical attack on PARM [19] and then we provide experimental results for this attack. Furthermore, we show that this attack can be prevented adopting PEAC. The attack scenario is the following:

- the sender generates authenticated messages using PARM and then sends these messages to all the receivers;
- an adversary eavesdrops the communication between the sender and the receivers, and can drop or delay chosen messages;
- the receivers validate the received messages using the PARM validation process.

The attack on PARM is built on an active adversary. The attacker has to store the broadcasted messages, preventing them from reaching the receivers and gathering enough evidence to forge a signature for his own message, which he will deliver to the receivers. Before of that, he has to deliver some intercepted messages in order to synchronize the usage tables of the receivers with his evidence.

In details, the adversary generates a message $M'$ and computes which elements of the TSK-chain it needs to authenticate such a message. Since messages authenticated by the sender reveal a subset of secret informations (each message carries $p$ elements of the TSK), an adversary can eavesdrop the messages generated by the sender until all of the $p$ elements of TSK needed to forge an authenticated packet are collected. We can express the attack in the balls and bins model. Let $n$ be the number of elements of a TSK of the TSK-chain; we model these $n$ elements with $n$ bins. Let each authentication element of a TSK as a ball. Let $p$ be the number of the bins selected by the adversary (i.e. bins are selected according to the authentication elements needed to authenticate a bogus message). We denote these $p$ bins as $B' = \{B'_1, \ldots, B'_p\}$. Let $r$ be the number of revealed authenticated messages; since each message carries $p$ authentication elements: $m = rp$ is the number of revealed authentication elements of the TSK-chain. We can model the disclosure of an authenticated message as throwing $p$ balls into $n$ bins. According to this model, an adversary can forge an authenticated packet if, after throwing the $m$ balls into the $n$ bins, no bin in $B'$ is empty.

**Theorem 2.** The probability $(Pr(\overline{A}))$ that the adversary can forge an authenticated packet for a message $M'$ (that is, no bin in $B'$ is empty) after $m = rp$ authentication elements are disclosed (that is, after $m$ balls have been thrown) is:

$$Pr(\overline{A}) \geq 1 - \exp\left(-\frac{m}{n} + \log(p)\right)$$

**Proof.** Let $A_i$ be the event “$B'_i \subseteq B'$ is empty after $m$ balls are thrown” ; $Pr(A_i) = \left(1 - \frac{1}{2}\right)^m$. Let $A$ be the event “at least one $B_i \in B'$ is empty after $m$ balls are thrown”:

$$Pr(A) = Pr(A_1 \lor \ldots \lor A_p) \leq p \left(1 - \frac{1}{2}\right)^m \leq \exp\left(-\frac{m}{n} + \log(p)\right)$$

Let the event $\overline{A}$ be “no bin $B_i \in B'$ is empty after that $m$ balls are thrown”:

$$Pr(\overline{A}) = 1 - Pr(A) \geq 1 - \exp\left(-\frac{m}{n} + \log(p)\right)$$

□

Figures 5(a) and 5(b) show the lower bound on the probability that an adversary is able to forge an authenticated message for the PARM mechanism (as provided by Theorem 2) with respect to the number of revealed authenticated messages, where parameters are set to $n = 1024$ and $n = 2048$ respectively. These results seem to suggest a stronger resilience of PARM with respect to PEAC. However, note that the derived bounds are just a lower bound on the probability to forge a signature. Figures 6(a) and 6(b) compare the analytical lower bound above derived with the experimental results. The experimental results show that PARM is more vulnerable than PEAC. Moreover, note that on one hand, once the TSK expires, in PARM the new TSK has to be sent to all receivers. On the other end, PEAC can just use longer hash chain (that is, increasing the parameter $L$) to amortize the overhead induced by the transmission of the TPK over several time intervals. Further, note that in PEAC the sender can decide to send less messages per time interval to achieve a certain level of security (e.g. $2^{-64}$ or $2^{-80}$ in our examples, but the security parameter can be completely customizable via the $n$ and $p$ parameters). Leveraging the length of the hash
Figure 3: Upper bound on the adversary’s capability of forging a PEAC signature.

Figure 4: Number of allowed revealed messages for probability of forging PEAC less than $2^{-64}$ (a) and $2^{-80}$ (b).

Figure 5: Lower bound on the adversary’s capability of forging a PARM signature.
chains (that is, the parameter \(L\)), no overhead is incurred by the receivers. That is, PEAC is completely customizable introducing almost no additional overhead on the receivers and just a slight overhead on the sender side (that is, the overhead required to manage longer hash chains).

### 5.3 Computation overhead

The computations required on the end-user side sum up to just \(p\) hashes per packet. As for the sender, note that when \(p\) authentication elements are required, at least \(p\) hash computation have to be performed. However, note that it is possible that one computed hash could collide with one of the previously generated hash (within the same time interval). In this case, the signer increments the counter and iterate the computation till no collision is verified. However, this increases the number of hash to be computed. If we assume that \(p/(n-m) \leq 1/2\) and \(n-m \leq n/2\) (that is, no more than \(n/2\) elements have already been disclosed, and \(p\) is less than half of the number of authentication elements that have not been assigned yet), than it can be shown that at most \(4p\) hash computations are required on average to generate \(p\) distinct authentication elements. Due to space limitation, we will delve with probabilistic analysis of this problem in the full version of this paper.

### 5.4 Communication and storage overhead

The bandwidth overhead induced by PEAC is of \(p(|\text{hash} + |c_a| + |T|)\) bits, where \(p\) is the number of authentication elements, \(|\text{hash}|\) is the hash function output size, \(|c_a|\) is the average bit length of a counter, and \(|T|\) is the size of the timestamp. We have proved in Section 5.3 that, on the average, if we assume that \(p/(n-m) \leq 1/2\) and \(n-m \leq n/2\), an overhead of at most \(4p\) hash has to be sustained on the average. Assuming to consume at most \(n/2\) elements per \(TSK\), the average value of the counter does not exceed the value \(n/2).\)

Hence, on the average \(|c_a| \leq \log((n/2)4p) = 1 + \log p + \log n\). For instance, for \(p = 10\), \(n = 2048\), \(|T| = 32\), and using MD5, the authentication element size is on the order of: \(p(|\text{hash}| + 1 + \log p + \log n + |T|) = 10(128+1+2+11+32) = 1740\) bits only.

Note that bandwidth overhead induced by TESLA is one HMAC (on the orde of 128 bits using MD5), in addition to one key disclosure per period of time (say, other 128 bits using AES). However, note that TESLA requires buffering, while our solutions provides instant message authentication.

As for storage, the receivers need to store the \(TPK\). Since the \(TPK\) can be composed of thousands of elements (e.g. up to \(2^{11}\) elements in our examples): the receivers need to store the entire \(TPK\) where each element is a hash-value (128 bytes using MD5). Thus, in our examples the public key can assume the value of 1024 and 2048 elements, requiring 128Kb and 256 Kb respectively. Note that the above storage requirements can be easily fulfilled by a wide range of receiving devices.

### 6. CONCLUDING REMARKS

In this paper we have introduced PEAC: a probabilistic, efficient, and resilient lightweight broadcast authentication protocol. Our protocol enjoys all the features required to an ideal broadcast authentication mechanism. In particular, PEAC is: efficient as for computations and memory required, has a small communication overhead, allows the receiver to authenticate each individual packet as soon as it is received (i.e. no buffering is required), provides perfect robustness to packet loss, and scales to a large number of receivers. Hence, it is particularly suitable for low-end resource constrained receivers and applications that require real-time or quasi-real time processing of the received packets.

We have shown that PEAC outperforms state of the art solution and that it is completely customizable, allowing to trade-off the provided security with a (small) overhead increase on the sender side only —that is just managing longer hash chains—, while not affecting the (small) overhead incurred by receivers.

### 7. ACKNOWLEDGMENTS

This work was partly supported by: The Spanish Ministry of Science and Education through projects TSI2007-65406-C03-01 “E-AEGIS” and CONSOLIDER CSD2007-00004 “ARES”, and by the Government of Catalonia under grant 2005 SGR 00446; The IST project SENSEI (Integrating the Physical with the Digital World of the Network of the Future, Grant Agreement Number: 215923).

### 8. REFERENCES


Figure 6: Lower bound on the adversary’s capability of forging a PARM signature.


