Emergent Response (ER): An Efficient and Scalable Real-time Broadcast Authentication Scheme for Command and Control Messages

Attila A. Yavuz, Member, IEEE

Abstract—Broadcast (multicast) authentication is crucial for large and distributed systems such as cyber-physical infrastructures (e.g., power-grid/smart-grid) and wireless networks (e.g., inter-vehicle networks, military ad-hoc networks). These time-critical systems require real-time authentication of command and control messages in a highly efficient, secure and scalable manner. However, existing solutions are either computationally costly (e.g., asymmetric cryptography) or unscalable/impractical (e.g., symmetric cryptography, one-time signatures, delayed key disclosure methods).

In this paper, we develop a new broadcast authentication scheme that we call Emergent Response (ER), which is suitable for time-critical authentication of command and control messages in large and distributed systems. We exploit the semi-structured nature of command and control messages to construct special digital signatures, which are computationally efficient both at the signer and verifier sides. We show that ER achieves several desirable properties that are not available in the existing alternatives simultaneously: (i) Fast signature generation and verification; (ii) immediate verification; (iii) constant size public key; (iv) compact authentication tag; (v) packet loss tolerance; (vi) being free from time synchronization requirement; (vii) provable security.

Index Terms—Secure Broadcast Authentication; Applied Cryptography; Security of Large Networked Systems; Network Security.

I. INTRODUCTION

A broadcast (multicast) authentication scheme enables each receiver in a large broadcast group to verify if the received message is intact and originated from the claimed sender. Broadcast authentication is a vital security service for many real-life applications (e.g., wireless networks [1]–[3]).

Secure broadcast authentication is a challenging problem, especially for large and distributed systems with time-critical applications [4]. For instance, cyber-physical infrastructure such as power-grid/smart-grid requires real-time authentication of control messages in an efficient, scalable and secure manner [5]. Other similar examples include vehicle-to-vehicle/infrastructure communication [6], disaster response systems (e.g., fire-sensors, earthquake warning) and military networks. In all these systems, immediate and secure authentication of command and control messages is essential to prevent adversaries from forcing catastrophic decisions by modifying or forging the messages [7]–[9].

The nature of such time-critical and distributed applications implies the following basic requirements on a broadcast authentication scheme: (i) Minimum end-to-end computation delay (i.e., fast signature generation/verification); (ii) immediate verification without message buffering to address the real-time nature of the applications; (iii) high scalability/applicability for large and distributed systems; (iv) small public key and signature size; (v) high packet loss tolerance; (vi) provable security without sacrificing the performance.

In Section I-A, we first revise the state-of-art solutions and identify their limitations in terms of the aforementioned requirements. We then present our contributions and highlight the desirable properties of our scheme in Section I-B.

A. Related Work

We present an overview of existing broadcast authentication techniques and discuss their advantages and limitations.

• Traditional Symmetric and Asymmetric Cryptography Methods: Symmetric cryptography based authentication methods rely on Message Authentication Code (MAC) [10] to achieve computational efficiency. Despite their simplicity, these methods are not practical to be used for broadcast (multicast) authentication purposes in large and distributed systems [4], [11]. That is, they require a pairwise key distribution between the signer(s) and verifiers to be secure, which makes them impractical even for moderately large systems. Moreover, due to their symmetric nature, these schemes are not publicly verifiable and therefore cannot achieve the non-repudiation property.

Traditional Public Key Cryptography (PKC)-based schemes (e.g., digital signatures) [10] rely on public key infrastructures. Hence, they are publicly verifiable, key compromise resilient and scalable for large systems. However, these schemes (e.g., RSA [12], ECDSA [13]) require Expensive Operations (ExpOps)\(^1\) such as modular exponentiation. These high computational costs make PKC-based schemes impractical for real-time (e.g., power-grid/smart-grid [7]) and resource-constrained applications.

• Delayed Key Disclosure Methods: The objective of delayed disclosure methods (e.g., TESLA [3] and its variants [14], [15]) is to achieve public verifiability of symmetric primitives while retaining their computational efficiency. Intuitively, a MAC is appended to every message and the key of

\(^1\)For brevity, we denote an expensive cryptographic operation such as modular exponentiation, elliptic curve scalar multiplication or cryptographic pairing as an ExpOp.
the MAC is disclosed in some subsequent packet. Disclosed keys are computed based on hash chains to increase the packet loss tolerance. Despite their advantages, these methods have the following drawbacks:

(i) Receivers cannot verify a message until its corresponding keying material is received (i.e., immediate verification is not possible). Such a delay is not tolerable for time-critical applications (e.g., power-grid/smart-grid). (ii) These schemes require a tight time synchronization between the sender and all receivers. Maintaining a continuous synchronization is challenging for large and distributed systems.

- **Signature Amortization Methods:** These methods (e.g., [16]–[19]) compute a digital signature over a set of messages instead of computing a distinct signature for each message. Hence, the cost of signature generation and verification is amortized over multiple messages.

However, signature amortization methods do not allow verification of an individual message in a given window until all related messages are received. Therefore, they cannot achieve the immediate verification. These schemes are also vulnerable to packet loss [20], since there is a correlation between different messages. Moreover, they still require ExpOps to compute and verify the signatures.

- **One-Time Signatures (OTSs):** OTSs achieve high computational efficiency and public verifiability since they rely on one-way functions without trapdoors. Preliminary OTSs [21] require very large private/public key and signature sizes. Later, more efficient constructions have been proposed (e.g., [22], [23]). One of the most efficient OTSs is the Hash-to-Obtain Random Subset (HORS) [23], which offers a fast signature generation/verification and relatively smaller signature size than its predecessors (e.g., BiBa [22]). Despite its elegance, HORS has the following limitations:

(i) A private/public key pair can be used only once without losing the security. It is also possible to compute a (small) limited number of signatures with the same private/public key by sacrificing the security (e.g., [22], [23]) and performance (e.g., [24], [25]). Therefore, all OTSs (including HORS) require distribution of new public keys, which causes heavy communication overhead. (ii) The public key/signature size of HORS is large and the certification (or chaining) of public keys incurs additional overhead. (iii) Various twists on HORS have been proposed to achieve different performance trade-offs (e.g., lower storage with the expense of very high computational cost [8], [26]) or security trade-offs (e.g., TV-HORS [7]). However, these schemes inherit the drawbacks of HORS. Moreover, they drastically decrease the efficiency of one parameter while only slightly increasing the efficiency of another parameter.

- **Online/offline Signatures:** The online/offline signatures shift expensive signature computations to the offline phase (these operations can be performed without knowing the actual message to be signed). This allows a fast signature generation in the online phase. Traditional online/offline signatures use OTSs as a building block. Hence, they inherit all the drawbacks of OTSs such as large public key and signature size. Recently, space efficient online/offline signatures have been proposed (e.g., [27]). However, they incur high computational overhead at the verifier side due to ExpOps.

In summary, existing methods are unable to meet the needs of time-critical broadcast applications.

### B. Our Contributions

In this paper, we develop a novel cryptographic scheme called **Emergent Response (ER),** which is especially suitable for broadcast authentication of command and control messages. We summarize the desirable properties of ER and compare it with the existing alternatives below:

- **High Signer and Verifier Computational Efficiency:** ER does not require any ExpOp in its online phase. That is, the total end-to-end computational overhead of ER is just a small number of multiplications and hash operations.

  - The computational efficiency of ER is at least an order of magnitude greater than traditional PKC-based signature schemes (e.g., [12], [13], [29], [30]).
  - ER is also much more computationally efficient and scalable than HORS variants (e.g., [7], [8], [26]) with security and performance trade-offs.

- **High Scalability and Communication Efficiency:** ER uses a single public key to verify (practically) unbounded number of signatures. Therefore, it is much more scalable and communication efficient than all OTS schemes (e.g., HORS [23] and other alternatives such as [7], [8], [22], [23], [26]), which require pre-distribution and retransmission of large size public keys. ER also avoids related problems such as certification or chaining of such public keys.

- **Compact Public Key and Signature:** The public key and signature size of our scheme is nearly the same with its building block scheme (i.e., RSA [12]). Hence, it is much more efficient than OTS schemes, which have large public key and signature size (e.g., [7], [21]–[23]).

- **Immediate Verification and being Free from the Time Synchronization Requirement:** ER does not rely on delayed key disclosure or message buffering. Therefore, unlike TESLA variants ([3], [15]) and signature amortization techniques (e.g., [20]), it can address real-time applications.

- **Individual Message Authentication and High Robustness:** In ER, receivers can verify each individual message received. Hence, ER is much more resilient against the packet loss than signature amortization techniques (e.g., [17], [20]).

Table I summarizes the high-level comparison of ER with its counterparts. A detailed performance analysis and comparison is given in Section V.

**Limitations:** ER requires that the broadcast entity is resourceful in order to store pre-computed message-signature tables. This requirement is reasonable for our envisioned applications, in which the signers are generally resourceful entities such as laptops, command centers or base stations.

Note that ER leverages the semi-structured nature of command and control messages to achieve its desirable properties. Hence, ER is not suitable for the applications, in which the message content is randomized and non-structured.

The remainder of this paper is organized as follows. Section II gives the definitions and ER system and security models. Section III describes ER in detail. Section IV gives a detailed...
security analysis of ER. Section V presents performance analysis and compares ER with previous approaches. Section VI concludes this paper.

II. DEFINITIONS AND MODELS

In this section, we first give the notation and definitions used by ER. We then provide the system and threat and security models of ER, respectively.

A. Notation and Definitions

Notation: Operators | and [x] denote the concatenation operation and the bit length of variable x, respectively. x \bar{x} S denotes that variable x is randomly and uniformly selected from set S. For any integer i, (x_0, \ldots, x_i) \bar{x} S means (x_0 \bar{x} S, \ldots, x_i \bar{x} S). |S| denotes the cardinality of set S. \{x_i\}_{i=0} denotes (x_0, \ldots, x_i). We denote by \{0,1\}^* the set of binary strings of any finite length. Full Domain Hash (FDH) (e.g., [31]) function H is defined as H : \{0,1\}^* \rightarrow Z_n^* and is modeled as an ideal hash function [32], where n is a large integer and Z_n^* is a multiplicative group (FDH can be easily derived from a standard hash function such as SHA-1 [33]).

ER is based on RSA [12] and condensed-RSA [34], which are defined below:

Definition 1 RSA signature scheme is a tuple of three algorithms (Kg, Sig, Ver) defined as follows:
- (sk, PK) \leftarrow RSA.Kg(1^c): The key generation algorithm takes the security parameter 1^c as the input. It randomly generates two large primes (p, q) and computes n = p \cdot q. The public and secret exponents (e, d) \in Z_n^* satisfies e \cdot d \equiv 1 \mod \phi(n), where \phi(n) = (p - 1)(q - 1). It returns a private/public key pair sk \leftarrow d and PK \leftarrow (n, e) as the output.
- \sigma \leftarrow RSA.Sig(sk, m): The signature generation algorithm takes sk and a message m \in \{0,1\}^* as the input. It returns a signature \sigma \leftarrow [H(m)]^d \mod n as the output (also denoted as \sigma \leftarrow RSA.Sig_{sk}(m)).
- c \leftarrow RSA.Ver(PK, m, \sigma): The signature verification algorithm takes PK, m and \sigma as the input (also denoted as c \leftarrow RSA.Ver_{PK}(m, \sigma)). If \sigma^e = H(m) \mod n it returns bit c = 1 meaning valid. Otherwise, it returns c = 0 meaning invalid.

Definition 2 Condensed-RSA scheme (denoted as C-RSA) is a tuple of three algorithms (Kg, Sig, Ver) defined as follows:
- (sk, PK) \leftarrow C-RSA.Kg(1^c): Execute RSA.Kg(1^c) in Definition 1.
- \sigma \leftarrow C-RSA.Sig(sk, \bar{m}): The signature generation algorithm takes sk and a set of messages \bar{m} = (m_0, \ldots, m_l) as the input. It returns a signature \sigma \leftarrow \prod_{j=0}^l \sigma_j^{m_j} \mod n as the output, where \sigma_j \leftarrow [H(m_j)]^{d_j} \mod n for j = 0, \ldots, l (also denoted as \sigma \leftarrow C-RSA.Sig_{sk}(\bar{m})).
- c \leftarrow C-RSA.Ver(PK, \bar{m}, \sigma): The signature verification algorithm takes PK, \bar{m} and \sigma as the input (also denoted as c \leftarrow C-RSA.Ver_{PK}(\bar{m}, \sigma)). If \sigma^e = \prod_{j=0}^l H(m_j) \mod n it returns bit c = 1 meaning valid. Otherwise, it returns c = 0 meaning invalid.

We give the cryptographic interfaces of ER below, which are used in our security model (i.e., Definition 4). The detailed description of ER algorithms is given in Section III.

Definition 3 ER is a tuple of four algorithms, (Kg, Offline-Sig, Online-Sig, Ver) defined as follows:
- (sk, PK) \leftarrow ER.Kg(1^c): The key generation algorithm takes the security parameter 1^c as the input. It returns a private/public key pair (sk, PK) as the output.
- sk \leftarrow ER.Offline-Sig(sk, \bar{M}): The offline signature generation algorithm is executed offline before the system
deployment. It takes the private key \( sk \) and a vector of pre-defined message sets \( \bar{M} = (M_0, \ldots, M_l) \) as the input, where each \( M_j \) is a set for \( j = 0, \ldots, l \). It returns a cryptographic token \( \bar{sk} = (\Gamma, \bar{\beta}) \) as the output, where \( \Gamma \) and \( \bar{\beta} \) denote pre-computed signature and message sets (defined in Section III).

- \( \sigma \leftarrow ER.Online-Sig(\bar{sk}, \bar{m}) \): The online signature generation algorithm takes the cryptographic token \( \bar{sk} \) (i.e., the private key of the online phase) and an online message \( \bar{m} = (m_0, \ldots, m_l) \) as the input. It returns a signature \( \sigma = (r, s) \) as the output, which is comprised of a random number \( r \) and a condensed-RSA signature \( s \).

- \( c \leftarrow ER.Ver(PK, \bar{m}, \sigma) \): The signature verification algorithm takes \( PK \), a message \( \bar{m} \) to be verified and its corresponding signature \( \sigma \) as the input. It outputs a bit \( c \), with \( c = 1 \) meaning \( valid \) and \( c = 0 \) meaning \( invalid \).

### B. System Model

Our system model relies on the PKC-based broadcast authentication model (e.g., [22], [23]), which includes two types of entities:

(i) A resourceful broadcast entity (i.e., the signer), which broadcasts message-signature pairs to the receivers. The signer is assumed to be trusted and it cannot be compromised by the adversary. This is compatible with our envisioned applications, in which the signer is storage capable and trusted such as a command center, base station or a satellite. (ii) Computational, storage and bandwidth limited receivers (e.g., verifiers). A verifier can be any (untrusted) entity (e.g., a sensor) and it might be compromised by adversary.

We assume that the signer executes the key generation and initial offline signature generation before the deployment. After the deployment, the signer can execute the offline signature generation either on-demand or periodically. Note that the offline phase must be performed independently from the online phase to achieve a high real-time computational efficiency.

### C. Threat and Security Model

Our threat model reflects how a PKC-based broadcast authentication scheme works. That is, adversary \( A \) can observe message-signature pairs computed under \( sk \). \( A \) also can actively intercept, modify, inject and replay messages transmitted over the network.

A standard security notion that captures our threat model is Existential Unforgeability under Chosen Message Attack (EU-CMA) [35].

\( \mathcal{E} \) is proven to be a EU-CMA signature scheme based on the experiment defined in Definition 4. In this experiment, \( \mathcal{A} \) is provided with a signing oracle \( ER.Online-Sig_{sk}(\cdot) \). \( \mathcal{A} \) can adaptively query \( ER.Online-Sig_{sk}(\cdot) \) on any message \( \bar{m} \) she wants up to \( L \) queries in total. The signing oracle returns the corresponding \( \mathcal{E} \) signature of \( \bar{m} \) under \( sk \). Finally, \( \mathcal{A} \) outputs a forgery \((\bar{m}^{*}, \sigma^{*}) \) under \( PK \). If this forgery is valid and non-trivial (i.e., \( \mathcal{A} \) did not query \( \sigma^{*} \) before), \( \mathcal{A} \) wins the EU-CMA experiment. Otherwise, \( \mathcal{A} \) loses in the EU-CMA experiment.

### Definition 4 EU-CMA experiment for \( \mathcal{E} \) is as follows:

\[
(\bar{m}, \sigma) \leftarrow \mathcal{E}.Kg(1^{\kappa}),
(\bar{m}^{*}, \sigma^{*}) \leftarrow \mathcal{A}^{ER.Online-Sig_{sk}(\cdot)}(PK),
\]

If \( \mathcal{E}.Ver(PK, \bar{m}^{*}, \sigma^{*}) = 1 \) and \( \bar{m}^{*} \) was not queried to the signing oracle, return 1, else it return 0.

\( \mathcal{A} \)’s advantage is

\[
Adv_{\mathcal{E}}^{EU-CMA}(\mathcal{A}) = Pr[\mathcal{E}.Ver(PK, \bar{m}^{*}, \sigma^{*}) = 1].\]

### Definition 5 EU-CMA experiment for C-RSA is as follows:

\[
(\bar{m}, \sigma) \leftarrow C-RSA^{(1^{\kappa})},
(\bar{m}^{*}, \sigma^{*}) \leftarrow \mathcal{A}^{C-RSA.Sign_{sk}(\cdot)}(PK),
\]

If \( C-RSA.Ver(PK, \bar{m}^{*}, \sigma^{*}) = 1 \) and \( \bar{m}^{*} \) was not queried to the signing oracle, return 1, else it return 0.

\( \mathcal{A} \)’s advantage is

\[
Adv_{\mathcal{E}}^{C-RSA}(\mathcal{A}) = Pr[\mathcal{E}.Ver(C-RSA, PK, \bar{m}^{*}, \sigma^{*}) = 1].\]

### III. Description of Our Scheme

In this section, we present our proposed scheme \( \mathcal{E} \). We first give an overview of \( \mathcal{E} \) and then provide its detailed description.

### A. Overview

In some DLP-based schemes (e.g., DSA with pre-computed tokens [28]), it is possible eliminate ExpOps from the online signature generation via pre-computed cryptographic tokens, which can be generated offline (independent from messages to be signed). This property enables a fast signature generation in the online phase. However, these schemes still require ExpOps for the signature verification, which makes them computationally costly. The online/offline signatures also have similar drawbacks, which have been discussed in Section I-A.

Some alternative signature schemes (e.g., RSA [12], Rabin [29]) are efficient at the verifier side (with a small public coefficient such as \( e = 3 \)) but they are costly at the signer side. In contrast to their DLP-based counterparts, these schemes do not offer a natural way to shift costly signature generation operations to the offline phase.

\( \mathcal{E} \) aims shifting costly signature generation operations of RSA to the offline phase, \( without \) requiring ExpOp at the verifier side and \( without \) inheriting the drawbacks of traditional OTSs (e.g., HORS [23]).

#### The Main Idea

We summarize the intuition behind of our scheme below:

1. **Semi-Structured Nature of Command and Control Messages**: We observe that the content of a command and control
message is generally structured in many real-life applications. That is, such a command and control message is semantically fragmented into subsections. A possible structure of such a message is comprised of a command, its intended receivers, a timestamp and some optional parameters (e.g., constants).

b) Pre-computation of Verifier Efficient Signatures (Offline Phase): ER exploits already existing structures in command and control messages to enable pre-computation for a verifier efficient scheme such as RSA [12]. In certain cases, the number of possible sub-messages in a given command and control message (i.e., commands and receiver groups) are limited. Hence, it is practical to pre-compute and store a RSA signature on each of those sub-message components, ER pre-computes sub-message/RSA-signature tables (i.e., $\beta$) during its offline phase (i.e., step 2-b of $\text{ER.OFFLINE-SIG}$ in Section III-B), which eliminates ExpOps from the online phase.

c) Efficient and Secure Combination (Online Phase): Assume that the signer needs to multicast a message $\vec{m} = (i$-th Command; j-th Receiver ; k-th Parameter) during the online phase. Firstly, the signer fetches the corresponding pre-computed signatures $(\text{sig}_i, \text{sig}_j, \text{sig}_k)$ from sub-message/signature table $\beta$. The signer then combines these signatures into a single and compact RSA signature as \( s \leftarrow \text{sig}_i \cdot \text{sig}_j \cdot \text{sig}_k \), which is a valid condensed-RSA [36] signature on message $\vec{m}$. Note that the online phase of ER is very efficient, since it requires only a few modular multiplications. However, this combination strategy requires further improvements as discussed below:

(i) Freshness and Dynamic Timestamping: Each message must be timestamped to prevent replay attacks. The signature of time stamp is generated via an another time value/signature table (similar to the sub-message/signature table). This prevents the signer from computing a costly online signature for the timestamp (i.e., step 2-b of $\text{ER.OFFLINE-SIG}$ in Section III-B). The optimization described in Remark 1 reduces the storage/computation overhead of these operations.

(ii) One-time Masking: Condensed-RSA is multiplicative homomorphic and mutable\(^2\) signature scheme [34]. Thus, if an adversary $A$ observes sufficient number of message/signature pairs $(m_j, s_j)_{j=0}^t$, she can recover the individual signatures stored in table $\beta$ (e.g., $\text{sig}_i, \text{sig}_j$ and $\text{sig}_k$ in this example) by solving multiplicative modular equations revealed via $(s_0, \ldots, s_t)$. $A$ then can create a valid condensed-RSA signature on any message. There are immutable signature techniques (e.g., [36]), which aim to prevent such an attack. However, these techniques are either interactive or computationally costly (e.g., [36]), which make them impractical.

We address this problem by developing a one-time signature masking technique, which does not require any ExpOp in the online phase. That is, the signer pre-computes a set of RSA signatures $\gamma_j$ on random numbers $r_j[\vec{\tau}]$. The signer stores these values in a table as $\Gamma = (r_j, \gamma_j)$ for $j = 0, \ldots, t'$, where $t'$ is the total number of random numbers (i.e., step 2-c of $\text{ER.OFFLINE-SIG}$ in Section III-B). In the online phase, the signer randomly picks a $(r, \gamma)$ pair from $\Gamma$ and masks the condensed RSA signature of $\vec{m}$ by multiplying it with $\gamma$ as $\sigma \leftarrow \gamma \cdot (s)$ (i.e., Equation 1 in $\text{ER.ONLINE-SIG}$ in Section III-B). Since $\gamma$ is a random number in $Z_n$, this operation one-time masks the individual signature components of $\beta$. This solves the aforementioned problem.

d) Scalable and Efficient Verification: The signature verification of ER is equivalent to a condensed-RSA signature verification of the signature component $s$ on message $\vec{m} = (r, \vec{m})$. Therefore, ER signature verification is as efficient as a condensed-RSA signature verification, which is very fast for properly selected parameters (e.g., $e = 3$). Similarly, ER uses a single permanent public key $PK$ to verify signatures. Therefore, it is much more scalable and practical than OTSs (e.g., HORS) requiring public key re-distribution.

B. Detailed Description

We give the detailed description of ER below.

1) $(sk, PK) \leftarrow \text{ER.KeyGen}(1^n)$: Generate $\tau \leftarrow \{0, 1\}^c$ and a RSA private/public key pair as $(sk', PK') \leftarrow \text{RSA.KeyGen}(1^n)$. Set ER private/public key pair as $sk \leftarrow sk'$ and $PK \leftarrow (PK', \tau)$, respectively.

2) $\vec{sk} \leftarrow \text{ER.OFFLINE-SIG}(sk, \vec{M})$: The offline signature generation algorithm takes a set of message components $\vec{M}$ and $sk$ as the input. It outputs a signature-message table $\vec{sk} = (\Gamma, \beta)$ as follows:

a) Message Components: $\vec{M} \leftarrow \{M_0, \ldots, M_{L-1}\}$ denotes the message components, where $L$ is the total number of message components.

The first component $M_0 = (T_0|\ldots|T_{k-1})$ denotes the time stamp, where $k$ is the total number of time stamp components in $M_0$. Each $T_{0 \leq i \leq k-1}$ is also comprised of a set of time values $(t_{i,j})_{i=0}^L$ for $i = 0, \ldots, k-1$ and $j = 0, \ldots, |T_i| - 1$. For instance, given a time format “yyyy|mm|dd|hh|mm|ss|ms”, we set $k = 7$ (i.e., there are seven time fields in $M_0$), $|T_i| = 12$ (i.e., the total number of months in $T_i$). Each remaining message component $M_{1 \leq i \leq L-1}$ is comprised of a set of messages $m_{i,j}$ for $i = 1, \ldots, L - 1$ and $j = 0, \ldots, |M_i| - 1$. For instance, $M_1$ is the set of commands and $M_2$ is the set of receivers (e.g., $m_{2,0}$ is the first receiver in $M_2$).

b) Compute Message-Signature Tables: Given $M_0$, compute a signature on each time value $t_{i,j}$ as $\pi_{i,j} \leftarrow \text{RSA.Sign}(sk, t_{i,j}, \vec{i}|\vec{j}|0) \text{ for } i = 0, \ldots, k-1 \text{ and } j = 0, \ldots, |T_{i,j}| - 1$. The corresponding signature table of $M_0$ is $\beta = \{\pi_{i,j}\}_{i=0}^k|j=0}^{|T_{i,j}| - 1}$. Given $m_{i,j} \in M_i$, compute a signature on $m_{i,j}$ as $s'_{i,j} \leftarrow \text{RSA.Sign}(sk, m_{i,j}, |i|) \text{ for } i = 1, \ldots, L - 1 \text{ and } j = 0, \ldots, |M_i| - 1$. The corresponding signature table of $M_i$ is $\beta_i = \{s'_{i,j}\}_{i=1}^L|j=0}^{|M_i| - 1}$.

c) Compute Random Number-Signature Table: Compute one-time masking signatures as $r_j \leftarrow \{0, 1\}^c$ and $\gamma_j \leftarrow \gamma \cdot (s)$ (i.e., Equation 1 in $\text{ER.ONLINE-SIG}$ in Section III-B). Since $\gamma$ is a random number in $Z_n$, this operation one-time masks the individual signature components of $\beta$. This solves the aforementioned problem.

\(^2\)Mutability refers that given a set of valid signatures, it is easy to derive new and valid aggregated signatures, which have not been queried before [36].
d) Assign the private key of online phase \( sk \leftarrow (\beta, \beta_1, \ldots, \beta_{L-1}) \), where \( \beta = (\beta_0 = \beta_1, \ldots, \beta_{L-1}) \).

3) \( \sigma \leftarrow \text{ER.Online-Sig}(\overline{\sigma}, m) \): During the online phase, assume that the signer needs to sign a message \( \overline{m} \in \overline{M} \), where \( \overline{m} = (m_0, \ldots, m_l) \) and \( m_0 \) is the current time \((0 < l < L - 1)\). Compute the online signature \( \sigma \) as follows:

Fetch the corresponding signatures of time components

\[
m_0 = (t_0|\ldots|t_{k-1}) \text{ from } \beta_0 \text{ as } (\overline{s}_0, \ldots, \overline{s}_{k-1}).
\]

Fetch the corresponding signatures of remaining message values

\( (m_1, \ldots, m_l) \) from \( \beta_1, \ldots, \beta_l \) as \( (s'_1, \ldots, s'_l) \). Last, randomly select a pair \((r, \gamma)\) from \( \Gamma \) and erase the selected pair from \( \Gamma \). The signature on \( \overline{m} \) is:

\[
s \leftarrow \gamma \cdot \prod_{j=0}^{k-1} \overline{s}_j \prod_{i=1}^{l} s'_i, \sigma \leftarrow (r, s)
\]

Individual signatures (i.e., \( \overline{s} \) and \( s' \)) of \( \overline{\beta} = \{\beta_0, \beta_1, \ldots, \beta_{L-1}\} \) are never released. That is, they are protected by being masked by one-time signature \( \gamma \).

4) \( c \leftarrow \text{ER.Ver}(PK, \overline{m}, \sigma) \): Assume that the verifier receives \( (\overline{m}, \sigma) \) on time \( t' \) from the signer. The verifier checks whether time stamp \( m_0 \) matches with \( t' \) and \(|r| = \kappa \) hold. If these conditions do not hold, the verifier rejects the signature (i.e., the signature is obsolete or the signature component \( r \) is not in the required range). Otherwise, given \( m_0 = (t_0|\ldots|t_{k-1}) \) and \( PK = (PK', \overline{\tau}) \), the verifier verifies \( \sigma = (r, s) \) as follows:

\[
m' \leftarrow H(r||s') \cdot \prod_{j=0}^{k-1} H(t_j||0) \prod_{i=1}^{l} H(m_i||i)
\]

\[
c \leftarrow \text{RSA.Ver}_{PK}(m', s)
\]

Theorem 1 \( \text{Adv}^{\text{EU-CMA}}_{\text{ER}}(t, L, \mu) \) is bounded as follows,

\[
\text{Adv}^{\text{EU-CMA}}_{\text{ER}}(t, L, \mu) \leq \text{Adv}^{\text{EU-CMA}}_{\text{RSA}}(t', L', \mu'),
\]

where \( t' = O(t) + L \cdot (\text{RNG} + \text{CSIG}) \), \( \mu' = \mu + L \cdot (2\kappa) \) and \( L' = L \).

Proof: Let simulator \( A \) be an \( \text{ER} \) attacker. We construct a \( C \)-RSA attacker \( F \) that uses \( A \) as a sub-routine. That is, we set \( (sk', PK') \leftarrow C \)-RSA.\( \text{Kg}(1^n) \) as defined in Definition 5 (i.e., \( \text{EU-CMA} \) experiment for \( C \)-RSA) and run the simulator \( F \) as follows:

\begin{itemize}
  \item \textbf{Algorithm } \( F^{C \text{-RSA}.\text{Sig}_{\text{adv}}(\cdot)}(PK') \):
    \begin{itemize}
      \item \textbf{Setup: } \( F \) generates \( \tau \leftarrow \{0, 1\}^n \) and sets the public key for \( \text{ER} \) as \( PK' = (PK, \tau) \) as in \( \text{ER}.\text{Kg} \). Note that \( F \) does not know the private key \( sk \) corresponding to \( PK \). However, \( F \) has an access to \( C \)-RSA oracle under \( sk' = sk \) and therefore he can simulate \( \text{ER} \) signatures via this oracle as follows:
      \begin{itemize}
        \item \textbf{Execute } \( A^{\text{ER}.\text{Online-Sig}_{\text{adv}}(\cdot)}(PK') \): \( F \) replies \( A \)'s queries and then check the result of her forgery as below.
      \end{itemize}
    \end{itemize}
  \item \textbf{- Queries: } \( A \) adaptively queries data items \( \overline{m}_j = (m_0, \ldots, m_l) \) of her choice to \( F \) for \( j = 0, \ldots, L - 1 \). For each query \( \overline{m}_j \), \( F \) generates a random number \( r_j \leftarrow \{0, 1\}^n \) and creates a message \( \overline{m}_j \leftarrow (r_j||\tau, \overline{m}_j) \). \( F \) then queries \( C \)-RSA oracle on \( \overline{m}_j \) and obtains the corresponding signature as \( s_j \leftarrow C \)-RSA.\( \text{sk}(\overline{m}_j) \). \( F \) sets the signature on \( \overline{m}_j \) as \( \sigma_j \leftarrow (r_j, s_j) \) and returns \( \sigma_j \) to \( A \) as the query answer.
  \item \textbf{- Forgery of } \( A \) : After the query phase, \( A \) outputs a forgery for \( PK \) as \( (\overline{m}^*, \langle \sigma^* \rangle = (r^*, s^*)) \). By Definition 4, \( A \) wins if the following conditions hold: \( \text{ER}.\text{Ver}(PK, \overline{m}^*, \sigma^*) = 1 \land \overline{m}^* \notin \mathcal{D} \). If these conditions hold, \( A \) wins the \( \text{EU-CMA} \) experiment for \( \text{ER} \) and returns 1. Otherwise, \( A \) loses in the experiment and returns 0.
  \item \textbf{- Forgery of } \( F \) : If \( A \) loses in the \( \text{EU-CMA} \) experiment for \( \text{ER} \) then \( F \) also loses in the \( \text{EU-CMA} \) experiment for \( C \)-RSA, and therefore \( F \) aborts and returns 0. Otherwise, \( F \) proceeds as follows:
    \begin{itemize}
      \item Given that \( A \)'s forgery is \( (\overline{m}^*, \langle \sigma^* \rangle = (r^*, s^*)) \), \( F \) sets \( \overline{m}^* \leftarrow (r^*, \overline{m}^*) \) and returns the forgery on \( PK' \) as \( (\overline{m}^*, \sigma^*) \). By Definition 5, \( F \) wins the \( \text{EU-CMA} \) experiment for \( C \)-RSA if the following conditions hold:
        \( C \)-RSA.\( PK', \overline{m}^*, s^* \) = 1 \land \((\overline{m}^* \notin \mathcal{D}) \lor (r^* \notin \mathcal{L})\). If these conditions hold, \( F \) wins the \( \text{EU-CMA} \) experiment (i.e., the forgery is valid and non-trivial) and returns 1. Otherwise, \( F \) loses the experiment and returns 0.
    \end{itemize}
\end{itemize}

Remark 2 \( \text{ER.Online-Sig} \) algorithm is equivalent to compute a condensed-RSA signature \( \overline{s} \) on a message \( \overline{m} = (r, \overline{m}) \). Similarly, one may verify that \( \text{ER.Ver} \) algorithm is equivalent to verify a condensed-RSA signature \( \overline{s} \) on \( \overline{m} = (r, \overline{m}) \).

IV. SECURITY ANALYSIS

We prove that \( \text{ER} \) is an \( \text{EU-CMA} \) signature scheme in Theorem 1 below. Note that in our proof, we omit terms that are negligible in terms of \( \kappa \).
C-RSA:
- **Abort1**: $\mathcal{F}$ does not abort due to $\mathcal{A}$’s queries. $\mathcal{F}$ simulates $\mathcal{A}$’s queries by expanding each ER query with a $2k$-bit randomness (i.e., $r \mid \pi$) and requesting the signature of $\tilde{m} \leftarrow (r \mid \tilde{m})$ from C-RSA oracle. Therefore, $\mathcal{F}$ aborts if and only if he cannot obtain a valid signature from C-RSA oracle, whose probability is negligible. Therefore, we conclude $Pr[\text{Abort1}] = 1$.
- **Forge**: $\mathcal{A}$ wins the EU-CMA experiment for ER.
  
  If $\mathcal{F}$ does not abort then $\mathcal{A}$ also does not abort, since $\mathcal{A}$’s view in this experiment is perfectly indistinguishable from her view in a real-system (see the indistinguishability argument below). Therefore, the probability that $\mathcal{F}$ does not abort and $\mathcal{A}$ wins the experiment is $Pr[\text{Forge} | \text{Abort1}] = Adv_{EU-CMA}^{ER}(t, L, \mu)$.
- **Abort2**: $\mathcal{F}$ does not abort during his forgery phase.
  
  The probability that $\mathcal{A}$ wins the experiment without querying $\mathcal{F}$ is negligible as it requires a random guess or finding a collision on $H$. This guarantees that, by Definition 5 and Remark 2, the forgery $(\tilde{m}^*, s^*)$ is valid and non-trivial. Therefore, we conclude $Pr[\text{Abort2} | \text{Abort1} \land \text{Forge}] = Pr[\text{Win} | \text{Abort1}] = 1$.
- **Win**: $\mathcal{F}$ wins the EU-CMA experiment for C-RSA, whose probability is denoted as $Pr[\text{Win}] = Adv_{EU-CMA}^{ER-CRSA}(t', L', \mu')$.
  
  This occurs if all of the above events happen. That is, $Pr[\text{Win}] = Pr[\text{Abort1}] Pr[\text{Forge} | \text{Abort1}] Pr[\text{Abort2} | \text{Abort1} \land \text{Forge}]$. The above equality implies that the EU-CMA advantage of ER is bounded by the EU-CMA advantage of C-RSA as follows:
  
  $Adv_{EU-CMA}^{ER}(t, L, \mu) \leq Adv_{EU-CMA}^{ER-CRSA}(t', L', \mu')$

**Execution Time Analysis:** The running time of $\mathcal{F}$ is that of $\mathcal{A}$ plus the time it takes to respond $L$ (in total) ER queries. Each ER query requires drawing a random number and requesting a condensed-RSA signature from C-RSA oracle, whose costs are denoted as RNG and CSIG, respectively. Hence, the approximate running time of $\mathcal{F}$ is $t' = O(t) + L \cdot (RNG + CSIG)$.

**Indistinguishability Argument:** The real-view of $\mathcal{A}$ is a vector $\bar{A}_{\text{real}} = \{PK, \sigma_j\}_{j=0}^l$, where the public key and signatures are computed by ER.Kg and ER.Online-Sig algorithms. The simulated view of $\mathcal{A}$ is also a vector $\bar{A}_{\text{sim}}$ and it is identical to $\bar{A}_{\text{real}}$, where the public key and signatures are computed as follows:

  (i) Given $PK = (PK', \pi)$, $\mathcal{F}$ directly takes the RSA public key $PK'$ as her input and then randomly generates $\pi$ as in ER.Kg algorithm.
  
  (ii) During the simulation, $\mathcal{F}$ obtains a condensed-RSA signature $s$ on $(r, \tilde{m})$ from C-RSA oracle and replies $\mathcal{A}$’s signature query with $\sigma = (r, s)$. The actual ER.Online-Sig and ER.Ver algorithms are also equivalent to generate and to verify a condensed-RSA signature $s$ on $\tilde{m} = (r, \tilde{m})$ (recall Remark 2). Therefore, all the answers of $\mathcal{F}$ for $\mathcal{A}$’s ER signature queries are valid and perfectly indistinguishable.

The above statements show that all variables in $\bar{A}_{\text{real}}$ and $\bar{A}_{\text{sim}}$ are computed identically. Hence, the joint probability distributions of $\bar{A}_{\text{real}}$ and $\bar{A}_{\text{sim}}$ are equal as $Pr[\bar{A}_{\text{real}} = \bar{A}] = Pr[\bar{A}_{\text{sim}} = \bar{A}]$ (i.e., perfectly indistinguishable).

V. PERFORMANCE ANALYSIS

In this section, we present the performance analysis of ER and compare it with previous schemes using the following criteria: (i) The computational overhead of signature generation and verification; (ii) storage and communication overhead depending on the size of signature and the size of public key. For each of these criteria, we first analyze ER and then provide a comparison of it with previous schemes. We also discuss some qualitative criteria such scalability and applicability based on the criteria (i)-(ii).

In computational overhead analysis, we focus on online (real-time) and end-to-end computational efficiency to evaluate the practicality of compared schemes for real-time applications. In storage and communication overhead analysis, we assume that public keys are generated and distributed before the system deployment.

We select a representative scheme(s) from each main group of schemes discussed in Section I-A. We select RSA [12] and ECDSA [13] as the verifier efficient and signer efficient PKC-based schemes, respectively. We select HORS [23] and its variant TV-HORS [7] as OTS schemes, since HORS is one of the fastest OTSs. We select the improved online/offline signature in [27], which is more scalable than previous online/offline schemes (e.g., [37]).

A. Computational Overhead

**Offline (non-real-time) Overhead:** The key generation cost of ER is a random number generation plus a single execution of RSA.Kg. The ER.Online-Sig requires $c \cdot (RSA.Sig)$ computations to prepare message-signature table $\beta$, where $c$ denotes the total number of pre-computed signatures in $\beta$. ER pre-computes a random number-signature table $\Gamma$, which requires $K \cdot (RSA.Sig + RNG)$ computations for $K$ messages ($\Gamma$ is computed as in Remark 1).

**Online (real-time) Overhead:** The essential computational overhead criteria for all compared schemes is the online (real-time) computational overhead. In ER, the signature generation cost is $l \cdot (Muln)$ for a message with $l$ sub-message components. The signature verification requires $l$ hash computations and $l$ modular multiplications plus a condensed-RSA signature verification (with $e = 3$). That is, the signature verification cost is $(l + e)Muln + l \cdot H$.

**Comparison:** To achieve the fastest real-time response, for each compared scheme, we shift the ExpOp(s) of signature generation to the offline phase (if the scheme enables this property). That is, we implement an improved version of ECDSA [28], which allows pre-computing and storing cryptographic tokens. These tokens enable ExpOp-free signature
Suggested parameters for HORS [23] are omitted. ECDSA is implemented with tokens at the signer to achieve an ExpOp-free signing. ECDSA signature omit the low-cost operations if there are ExpOps in the sum (e.g., \(\text{Expn} n\)). TV-HORS and RSA/condensed-RSA, which is also used as a part of the public key of ER, \(u, \alpha\) and \(l\) denote the constant parameters used in HORS, TV-HORS and ER, respectively. End-to-end computational overhead is the sum of signer and verifier computational overhead, in which we omit the low-cost operations if there are ExpOps in the sum (e.g., \(\text{Expn}\)). Similarly, small and constant number of additions (e.g., as in ECDSA) are omitted. ECDSA is implemented with tokens [28] at the signer to achieve an ExpOp-free signing. ECDSA signature verification is performed with the double-point scalar multiplication.

### Table II

#### Analytical Online (i.e., Real-Time) Computational Cost Comparison of ER and Previous Schemes

<table>
<thead>
<tr>
<th></th>
<th>ER</th>
<th>Traditional PKC</th>
<th>Online/Offline [27]</th>
<th>(K pre-computed private/public key pairs)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signer</strong></td>
<td>(t \cdot \text{Muln})</td>
<td>(\text{Expn})</td>
<td>(H + 2\text{Mul})</td>
<td>(0.1 \cdot \text{Muln})</td>
</tr>
<tr>
<td><strong>Verifier</strong></td>
<td>((l+e)\text{Muln} + i \cdot H)</td>
<td>(e \cdot \text{Muln})</td>
<td>(\approx 1.3 \cdot \text{EMul})</td>
<td>(\text{Expn} + \text{Muln})</td>
</tr>
<tr>
<td><strong>End-to-end</strong></td>
<td>((2l + e)\text{Muln} + i \cdot H)</td>
<td>(\approx \text{Expn})</td>
<td>(\approx 1.3 \cdot \text{EMul})</td>
<td>(\approx \text{Expn})</td>
</tr>
</tbody>
</table>

### Table III

#### Average Online Execution Time (in \(\mu s\)) Comparison of ER and Previous Schemes (Sampled over \(K = 10^4\) Messages)

<table>
<thead>
<tr>
<th></th>
<th>ER(^*)</th>
<th>Traditional PKC</th>
<th>Online/Offline [27]</th>
<th>(K pre-computed private/public key pairs)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signer</strong></td>
<td>60</td>
<td>3768</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Verifier</strong></td>
<td>86</td>
<td>26</td>
<td>1550</td>
<td>20</td>
</tr>
<tr>
<td><strong>End-to-end</strong></td>
<td>146</td>
<td>3792</td>
<td>1576</td>
<td>21</td>
</tr>
</tbody>
</table>

ER\(^*\) and HORS\(^*\) are the most computationally efficient schemes among these alternatives. Note that despite being efficient, HORS is impractical as it requires distributing a large public key (e.g., 3KB-5KB) for each message. Table I and Section I-B present a qualitative comparison of ER and HORS. Section V-B presents a storage and communication overhead comparison of ER and HORS.

### Table IV

#### Analytical (Asymptotic) Storage and Communication Overhead Comparison of ER and Previous Schemes

<table>
<thead>
<tr>
<th></th>
<th>ER</th>
<th>Traditional PKC</th>
<th>Online/Offline [27]</th>
<th>(K pre-computed private/public key pairs)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signer</strong></td>
<td>(</td>
<td>n</td>
<td>+ (</td>
<td>n</td>
</tr>
<tr>
<td><strong>Verifier</strong></td>
<td>(</td>
<td>n</td>
<td>+ 2c)</td>
<td>(</td>
</tr>
<tr>
<td><strong>Communication</strong></td>
<td>(</td>
<td>n</td>
<td>+ \kappa)</td>
<td>(</td>
</tr>
</tbody>
</table>

Suggested parameters/bit length for TV-HORS [7] are \((t' = 12, v = 5, |H'| = 48)\) with \(\kappa = 54\), where \(|H'|\) is the truncated hash output. Suggested parameters for HORS [23] are \((t = 256, u = 20)\), where \(|H| = 80\). Bit lengths of \(|n|\), \(|p'|\) and \(|q'|\) are given as in Table III. \(c\) denotes the total number of pre-computed signatures in \(\beta\) for ER (with the optimization given in Remark 1). In this example, \(c = 10^3\).

### Table V

#### Numerical Storage and Communication Overhead Comparison of ER and Previous Schemes for \(K = 10^4\) Messages

<table>
<thead>
<tr>
<th></th>
<th>ER</th>
<th>Traditional PKC</th>
<th>Online/Offline [27]</th>
<th>(K pre-computed private/public key pairs)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signer</strong></td>
<td>260 KB</td>
<td>128 byte</td>
<td>101 KB</td>
<td>250 KB</td>
</tr>
<tr>
<td><strong>Verifier</strong></td>
<td>138 byte</td>
<td>128 byte</td>
<td>84 byte</td>
<td>256 byte</td>
</tr>
<tr>
<td><strong>Communication</strong></td>
<td>138 byte</td>
<td>128 byte</td>
<td>84 byte</td>
<td>256 byte</td>
</tr>
</tbody>
</table>

(i) After the corresponding keying material for \(K = 10^4\) messages depletes, HORS and TV-HORS require distributing \(K\) new public keys. Other schemes do not require such a public key re-distribution.

(ii) To achieve the minimum end-to-end delay, all compared schemes (with the exception of plain RSA) are analyzed based on the pre-computation/pre-distribution setting. That is, all tokens/public keys are pre-computed and distributed before the system deployment. Hence, ECDSA with tokens achieves a higher computational efficiency with the cost of a higher signer storage. A similar principle also applies to HORS and TV-HORS.

* Token pre-computation is not known for RSA. Hence, we implement it with the traditional single private/public key setting.
generation in the online phase. In HORS and TV-HORS, we generate $K$ one-time private/public key pairs in the offline phase. These private/public keys are stored at the signer and verifier sides, respectively (for immediate use in the online phase). Online/offline signatures (e.g., [27]) by nature realize such a pre-computation at the signer side. Note that, different from its DLP-based counterparts (e.g., ECDSA), such a pre-computation is not known for RSA.

Table II summarizes the analytical comparison of ER with its counterparts. We also prototype all compared schemes on a computer with an Intel(R) Core(TM) i7 Q720 at 1.60GHz CPU and 2GB RAM running Ubuntu 10.10. We measure the execution times using MIRACL [38] library. Table III provides an average (online) execution time comparison for all compared schemes.

We select an example message structure $\vec{m} = (i$-th Command; all receivers; hour; min; sec; ms), which includes $l = 6$ components (the signature of other time components are included in $\gamma$ as stated in Remark 1). Given that $(l = 6, e = 3)$, end-to-end delay of ER is as low as 146 $\mu$s (without any modular arithmetic optimization). ER is approximately 26, 12 and 13 times faster than RSA, ECDSA and online/offline scheme [27], respectively. ER is also much faster than TV-HORS but less efficient than HORS.

**Remark 3** Despite its computational efficiency, HORS requires distributing a new public key for per-message. Since the size of a HORS public key is very large (e.g., 3KB-5KB), this incurs intolerable communication and storage overhead. HORS also requires the certification and verification of these public keys. All these requirements make HORS impractical for large and distributed systems.

### B. Storage and Communication Overhead

In ER, the signer storage overhead is comprised of pre-computed tables $|\vec{\beta}|$, $|\Gamma|$. The size of $(|\vec{\beta}|, |\Gamma|)$ is $|n|c + (|n| + $κ$)|O(K)$, where $c$ denotes the total number of pre-computed signatures stored in $\vec{\beta}$. Based on the optimization technique described in Remark 1, the signer pre-computes a new $\Gamma$ from time to time. The size of $\Gamma$ is $(|n| + $κ$)|O(K)$ for $K$ messages to be signed during the designated time interval. Unlike OTS schemes (e.g., HORS [23]), ER does not require distributing public keys once these random numbers deplete. The signer just needs to compute a new $\Gamma$ in an offline manner (independent from the online computations).

The verifier storage overhead of ER is a constant size public key $PK$, which is comprised of a condensed-RSA public key $pk'$ and a $κ$-bit random number $Π$.

The communication overhead of ER is $|n| + |\kappa|$ (i.e., the size of signature $\sigma$).

**Comparison:** Table IV summarizes the analytical storage and communication overhead comparison of ER with its counterparts. Table V provides an example of some numerical values for $K = 10^4$ messages, with proper parameters in the pre-computation/pre-distribution setting. That is, all keys/tokens [28] are pre-computed and stored for ECDSA, HORS and TV-HORS. ER and online/offline scheme [27] already operate in this setting.

At the verifier side, ER is much more efficient than HORS and TV-HORS and more efficient than online/offline scheme [27]. However, it is less efficient than ECDSA and slightly less efficient than RSA. At the signer side, ER is more efficient than HORS but less efficient than ECDSA and TV-HORS.

### VI. Conclusion

In this paper, we developed a new broadcast authentication scheme for command and control messages, which we refer to as Emergent Response (ER). ER simultaneously achieves several desirable properties including fast signature generation and verification, immediate verification without message buffering, small public key and signature size, high scalability, high packet loss tolerance, provable security and being free from synchronization requirement. Our comparison with the existing alternatives shows that ER is an ideal choice for broadcast authentication of command and control messages in large and distributed systems with time-critical applications.

### REFERENCES


[38] Shamus, “Multiprecision integer and rational arithmetic c/c++ library (MIRACL),” http://www.shamus.ie/.