Compact Energy and Delay-Aware Authentication*

Muslum Ozgur Ozmen†‡ Rouzbeh Behnia†‡ Attila A. Yavuz§

Abstract

Authentication and integrity are fundamental security services that are critical for any viable system. However, some of the emerging systems (e.g., smart grids, aerial drones) are delay-sensitive, and therefore their safe and reliable operation requires delay-aware authentication mechanisms. Unfortunately, the current state-of-the-art authentication mechanisms either incur heavy computations or lack scalability for such large and distributed systems. Hence, there is a crucial need for digital signature schemes that can satisfy the requirements of delay-aware applications.

In this paper, we propose a new digital signature scheme that we refer to as *Compact Energy and Delay-aware Authentication* (*CEDA*). In *CEDA*, signature generation and verification only require a small-constant number of multiplications and Pseudo Random Function (PRF) calls. Therefore, it achieves the lowest end-to-end delay among its counterparts. Our implementation results on an ARM processor and commodity hardware show that *CEDA* has the most efficient signature generation on both platforms, while offering a fast signature verification. Among its delay-aware counterparts, *CEDA* has a smaller private key with a constant-size signature. All these advantages are achieved with the cost of a larger public key. This is a highly favorable trade-off for applications wherein the verifier is not memory-limited. We open-sourced our implementation of *CEDA* to enable its broad testing and adaptation.

Keywords: Applied cryptography, delay-aware authentication, real-time networks, digital signatures.

1 Introduction

Broadcast authentication is an essential security service for various important systems, where the authenticity of messages should be verified by multiple receivers. However, broadcast authentication is a challenging problem for large and distributed systems (e.g. smart grids, vehicular networks, IoT systems), especially if the system has real-time authentication requirements [21]. For instance, as mentioned in relevant vehicular network standards (e.g., [2, 18]), a single car might broadcast a very large number of messages (e.g., up to 500-1000 messages) per second, where all these messages should be verified by other vehicles/devices in the vicinity. Such messages may include directives for sudden brakes/turns, which require the timely reaction of the receiving parties. This also brings scalability problems since a vehicular network might be composed of a large number of components (e.g., vehicles, infrastructure, devices). Similarly, in power grid/smart grid systems, some critical command and

*This work is supported by the NSF CAREER Award CNS-1652389.

†Oregon State University, Corvallis, Oregon, USA, {ozmenmu,behniar}@oregonstate.edu

‡The first and second authors contributed equally to this work.

§University of South Florida, attilaayavuz@usf.edu. Work done in part while Attila A. Yavuz was at Oregon State University.

Table 1: Experimental Performance Comparison of *CEDA* and its Counterparts on ARM Cortex A53

Scheme	Signer		Transmission	Verifier		End-to-End Delay (μs)
	Signature Generation (μs)	Private Key (Byte)	Signature Size (Byte)	Signature Verification (μs)	Public Key (Byte)	
RSA	237.15	768	416	1.33	384	238.48
ECDSA	9.33	32	64	12.10	32	21.43
BPV-ECDSA	1.75	10272	64	12.10	32	13.85
Ed25519	2.25	32	64	6.79	32	9.04
SCRA-C-RSA	2.72	2000000	432	4.42	384	7.14
CEDA	**1.59**	416	416	**2.98**	393600	**4.57**

control messages must be verified by a large number of peripheral devices [38, 33] in real-time. Besides such real-time applications, an efficient authentication mechanism is also greatly needed by recently emerging IoT applications that involve resource-limited devices (e.g., small aerial drones).

1.1 Problem Statement

The current state-of-the-art authentication mechanisms might not be able to fully meet the demands of large and distributed time-critical applications (e.g., smart-grid, vehicular/drone networks). That is, Message Authentication Codes (MACs) are highly efficient but they lack the necessary scalability for large and distributed systems as well as public verifiability and non-repudiation properties. Digital signature schemes rely on public key infrastructures, and therefore can enable scalable authentication for large-distributed systems. However, unlike MACs, they generally require highly expensive operations at the signer's and/or verifier's side. For instance, standard signatures (e.g., RSA [30], ECDSA [4]) require expensive operations such as exponentiation or elliptic curve scalar multiplications, which have been shown to be highly costly for some delay-aware applications (e.g., smart-grid [1, 35, 36], vehicular networks [23, 39, 18, 2]).

Delay-aware signatures such as SCRA [37] and RA [36] were proposed, however both of these schemes incur very large private keys due to the pre-computation tables at the signer's side. Moreover, RA requires messages to have specific pre-defined structures, which might not be the case for various real-life applications. One-time signatures [29] and some of their variants (e.g., [35, 20]) offer very fast signature generation and verification, however they have large signature sizes. Moreover, the private-public key pair must be continuously renewed, whose overhead may not be practical for certain applications. Signature schemes that incur linear token/key storage (e.g. online/offline signatures [16]) are also not suitable for applications with memory-limited devices. Efficient signature generation and verification can be achieved by delayed key disclosure methods [26] and amortized signatures [22]. However, these methods rely on packet buffering, and therefore, highly intolerant to packet losses. Moreover, they lack the immediate verification critically required by delay-aware applications. In Section 2, we provide a detailed overview of the related works that are most relevant to ours.

There is a significant need for a compact and light-weight digital signature scheme that can achieve high-speed signature generation and verification for time-critical systems.

1.2 Our Contributions

In this paper, we developed a new real-time digital signature scheme that we refer to as *Compact Energy and Delay-aware Authentication* (*CEDA*). We summarize the desirable properties of *CEDA* as

follows (Table 1 demonstrates the experimental comparison between $CEDA$ and its counterparts on ARM Cortex A53).

- *Fast Signing:* The signature generation of $CEDA$ does not require any expensive operation such as exponentiation over large integers or elliptic curve scalar multiplication. More specifically, the signing algorithm in $CEDA$ only requires an exponentiation over a small modulus and cryptographic hash function calls that makes it the fastest among its counterparts. For instance, as shown in Tables 1 and 4, $CEDA$ can generate up to 18,070 and 628 signatures per second on a commodity hardware and IoT device, respectively.

- *Low End-to-end Delay:* $CEDA$ enjoys from the fastest signing algorithm and the second fastest verification algorithm among its counterparts. That is, the verification only requires an exponentiation over a small modulus and a few multiplications. More specifically, as shown in Table 1, $CEDA$ is 1.56$\times$ faster than its most efficient counterpart (SCRA in [37]) and 4.69$\times$ faster than ECDSA, in terms of end-to-end delay.

- *Eliminate the Pre-computation Components from Signer:* Some applications (e.g. IoT, smart-grids) may require memory-limited devices to issue signatures. Unlike some existing alternatives (e.g., [11, 37, 36, 13, 31, 16]), $CEDA$ does not require any pre-computation table or tokens to be stored at signer's side. For instance, SCRA-C-RSA [37] and ECDSA with pre-computation [11] require storing a private key of size 2 MB and 10 KB on the signer's side, respectively. $CEDA$ has a constant private key of size 416 Byte that is smaller than traditional RSA signature [30] and a signature size identical to the traditional RSA (see Table 1).

- *Immediate Verification:* Unlike some broadcast authentication mechanisms (e.g. [26]), $CEDA$ can achieve immediate verification without the need of packet buffering or time synchronization.

• *Limitation*: The main limitation of $CEDA$ is its large public key size (e.g., 393 KB for $\kappa = 128$-bit security) compared to its alternatives. However, in many delay-aware applications (e.g., aerial drones, vehicular networks, smart-grid), the verifying devices (e.g., cars, UAVs, command centers) are potentially more than capable of storing such public keys. Therefore, by providing the lowest end-to-end cryptographic delay with small private key sizes, $CEDA$ is expected to offer an ideal choice for time-critical networks, in which a very high-speed authentication is a crucial requirement to ensure a safe and reliable operation.

2 Related Work

In this section, we provide an overview of efficient digital signature schemes and authentication mechanisms that are most relevant to our work.

Standard Digital Signatures: Standard signatures (e.g., RSA [30], ECDSA [4]) require *expensive operations*, such as exponentiation over a large modulus, and elliptic curve scalar multiplication. Hence, they are not suitable for resource-limited devices and time-critical applications. Improvement via special elliptic curves [10] and/or pre-computation techniques [11] are possible. However, such improvements may not fully meet the demands of highly time-critical applications (see Section 6 for detailed analysis).

Delay-Aware Digital Signatures: Real-time signatures, specially designed for smart grids and vehicular networks were proposed in [37, 36]. Such schemes provide fast signature generation and verification to meet the requirements of time-critical networks. However, RA [36] relies on a pre-defined structure of messages, which may not be applicable for many real-life scenarios. Moreover, both of these signature schemes require large private key sizes (up to 2MB [37]), that may not be feasible for many resource-limited signers.

One-time Signatures (OTS) and Their Extensions: Hash-based signatures achieve post-quantum security [9]. Earlier one-time hash-based signatures (e.g., HORS [29]) offer fast signing and verification but have very large signature sizes (e.g., 2-5 KB). Moreover, a private/public key pair can only be used once and therefore, must be renewed frequently. This continuous renewal requires the distribution of new public keys and may be impractical for real-life applications where each new public key should be signed by a certificate authority and verified by the verifier. Different performance and security trade-offs, such as low storage with very high computational cost [20] and time valid OTS such as TV-HORS [35], have been offered based on HORS. Despite their benefits, time-valid approaches suffer from performance and security penalties due to time-synchronization requirements and low tolerance for packet loss. Moreover, the use of low-security parameters might not be ideal for some security-critical delay-aware applications even with potential time constraints. Multiple-time hash-based signatures (e.g., [8]) rely on Merkle-trees [24] with a signer state [12] to be able to sign several messages. Recently, stateless signatures (e.g., SPHINCS [9]) have been proposed. However, these schemes have extremely large signatures (up to 41 KB) and expensive signing algorithms for low-end devices [19].

Online/Offline Signatures: Online/offline signatures (e.g. [31, 16, 25]) pre-compute a token for each message to be signed at the offline phase, and then use it to compute a signature on a message efficiently at the online phase. However, these schemes can use a private/public key pair only once, and therefore introduce a linear public key size. Hence, all such online/offline signatures incur linear storage with respect to the number of messages to be signed, which might not be practical for resource-limited devices. Moreover, the tokens must be renewed continuously as depleted, which introduces further computational overhead. Therefore, they may not be practical for real-time networks or IoT devices as considered in this work.

Delayed Key Disclosure and Amortized Signatures: Delayed key disclosure methods [26] introduce an asymmetry between the signer and the verifier via a time factor, and therefore can achieve highly efficient signing and verification via only Message Authentication Codes. However, they require time synchronization among entities, packet buffering, and introduce potential packet loss risks. Therefore, such schemes cannot provide immediate verification, which is a critical requirement for real-time networks. Similarly, achieving time synchronization for a large distributed system might be difficult. In signature amortization techniques (e.g., [22]), the signer generates a signature over a set of messages to reduce the cost. However, this also requires packet buffering and introduces potential packet loss risks. Moreover, amortized signatures require all related messages in a single set to be received until a message could be verified, and therefore they lack immediate verification.

3 Preliminaries

We first outline the notation in Table 2 and then describe our building blocks.

Definition 1. A digital signature scheme is a tuple of three algorithms $\mathtt{SGN} = (\mathtt{Kg}, \mathtt{Sig}, \mathtt{Ver})$

Table 2: Notation followed to describe schemes.

(t, k)	HORS parameters (k out of t)
κ	Security parameter
N	RSA modulus
p, q	large primes
d	RSA large exponent
e	RSA small exponent
z	*CEDA* private key
s_i	Random components generated deterministically by z
V_i	*CEDA* public key
r	One-time randomness
c	Counter
PRF	Pseudo Random Function
PRF_1¶	$PRF_1 : \{0,1\}^* \rightarrow \{0,1\}^\kappa$
PRF_2¶	$PRF_2 : \{0,1\}^* \rightarrow \{0,1\}^\kappa$
H	Cryptographic hash function
H_1	$H_1 : \{0,1\}^* \rightarrow \{0,1\}^{l_1}$ where $l_1 = 2 \cdot \kappa$
H_2	$H_2 : \{0,1\}^* \rightarrow \{0,1\}^{l_2}$ where $l_2 = k \cdot \log_2 t$

¶ PRF_1 and PRF_2 are two different PRF instantiations with the same domain.

defined as follows.

- $(sk, PK) \leftarrow \mathtt{SGN.Kg}(1^\kappa)$: Given the security parameter κ, this algorithm outputs the private/public key pair (sk, PK).
- $\sigma \leftarrow \mathtt{SGN.Sig}(m, sk)$: Given a message to be signed m and the private key of the signer (sk), this algorithm outputs the signature σ.
- $\{0,1\} \leftarrow \mathtt{SGN.Ver}(m, \sigma, PK)$: Given a message-signature pair to be verified (m, σ), and the public key of the signer (PK), this algorithm outputs a bit that indicates if the signature is verified (1) or not (0).

Definition 2. Existential Unforgeability under Chosen Message Attack (EU-CMA) experiment $Expt^{EU-CMA}_{\mathtt{SGN}}$ is defined as follows.

- $(sk, PK) \leftarrow \mathtt{SGN.Kg}(1^\kappa)$
- $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathtt{SGN.Sig}(\cdot)}(PK)$
- If $1 \leftarrow \mathtt{SGN.Ver}(m^*, \sigma^*, PK)$ and m^* was not queried to $\mathtt{SGN.Sig}(\cdot)$, return 1, else, return 0.

The EMU-CMA advantage of $\mathcal{A}$ is defined as $Adv^{\text{EU-CMA}}_{\text{SGN}} = \Pr[Expt^{EU-CMA}_{\text{SGN}} = 1]$.

Given a one-way function f, HORS signature scheme is defined in the following definition.

Definition 3. HORS signature scheme consists of three algorithms $\text{HORS} = (\mathtt{Kg}, \mathtt{Sig}, \mathtt{Ver})$ defined as follow.

- $(sk, PK) \leftarrow \text{HORS}.\text{Kg}(l, k, t)$: Given parameters l, k and t, this algorithm generates t random l-bit strings $(s_1, s_2 \ldots, s_t)$, computes $v_i = f(s_i)$ for $1 \leq i \leq t$ and outputs $sk = (s_1, s_2 \ldots, s_t)$ and $PK = (v_1, v_2 \ldots, v_t)$.
- $\sigma \leftarrow \text{HORS}.\texttt{Sig}(m, sk)$: Given a message m to be signed, this algorithm computes $h = H(m)$ and splits h into k substrings $(h_1, h_2, \ldots, h_k)$, each of length $\log_2 t$. The substrings are interpreted as integers i_j for $1 \leq j \leq k$ and used to generate a signature as $\sigma = (s_{i_1}, s_{i_2}, \ldots, s_{i_k})$.
- $\{0, 1\} \leftarrow \text{HORS}.\texttt{Ver}(m, \sigma, PK)$: Given a message-signature pair $(m, \sigma = (s_{i_1}^{'}, s_{i_2}^{'}, \ldots, s_{i_k}^{'}))$, this algorithm computes $h = H(m)$ and splits h into k substrings $(h_1, h_2, \ldots, h_k)$. The substrings are interpreted as integers i_j for $1 \leq j \leq k$. Returns 1 if for each j, $f(s_j^{'}) = v_{i_j}$ and returns 0 otherwise.

Definition 4. A trapdoor permutation function family is a tuple of algorithms $\pi = (\texttt{Gen}, \texttt{Eval}, \texttt{Invert})$ as follows.

- $(i, td) \leftarrow \pi.\texttt{Gen}(1^\kappa)$: Given the security parameter κ, this algorithm outputs a pair (i, td), where i is the index of a particular permutation π_i defined over some domain D_i, and td is the trapdoor that allows for the inversion of π_i.
- $y \leftarrow \pi.\texttt{Eval}(i, x)$: Given an index i and $x \in D_i$, this algorithm outputs an element $y \in D_i$. More specifically, for all i output by $\texttt{Gen}$, the function $\texttt{Eval}(i, \cdot) : D_i \rightarrow D_i$ is a permutation.
- $x \leftarrow \pi.\texttt{Invert}(td, y)$: Given a trapdoor td and y, this algorithm outputs the element $x \in D_i$.

The correctness of a trapdoor permutation family requires that for all κ, all (i, td) output by $\texttt{Gen}$, and all $x \in D_i$, we have $x \leftarrow \texttt{Invert}(td, y)$.

Definition 5. An RSA permutation function is defined as a tuple $RSA = (\texttt{Gen}_{\text{RSA}}, \texttt{Eval}_{\text{RSA}}, \texttt{Invert}_{\text{RSA}})$ as below.

- $\langle (N, e), (N, d) \rangle \leftarrow \texttt{Gen}_{\text{RSA}}(1^\kappa)$: Given the security parameter κ, it chooses two large primes p and q and forms their product $N \leftarrow p \cdot q$. It then computes $\phi(N) \leftarrow (p-1) \cdot (q-1)$, chooses e that is relatively prime to $\phi(N)$ and computes d where $e \cdot d \equiv 1 \mod \phi(N)$. It outputs (N, e) as the index i, and (N, d) as the trapdoor td. The domain $D_{N,e}$ is $\mathbb{Z}_N^*$.
- $y \leftarrow \texttt{Eval}_{\text{RSA}}((N, e), x)$: Given the index (N, e) and a random element $x \in \mathbb{Z}_N^*$, this algorithm computes and outputs $y \leftarrow x^e \mod N$.
- $x \leftarrow \texttt{Invert}_{\text{RSA}}((N, d), y)$: Given (N, d) and an element y, it computes the inversion as $x \leftarrow y^d \mod N$.

Definition 6. Inverting the RSA permutation function defined in Definition 5 without having the knowledge of the trapdoor information td is known to be a hard problem [3]. Given, a public key (N, e) and $x \in \mathbb{Z}_q$ the advantage of the adversary $\mathcal{A}$ is defined as $Adv_{RSA} = \Pr[y \leftarrow \texttt{Eval}_{\text{RSA}}((N, e), x); x \leftarrow \mathcal{A}_{(\kappa, N, e)}(y)] < \epsilon$.

• **Security and System Model**: The standard security notion that captures our threat model is EU-CMA as in Definition 2

Our system model is based on Public Key Cryptography broadcast authentication model which includes two types of entities (i.e., the signer and the verifier). As depicted in Figure 1, we assume that a key generation server, uploads the private key to the signer (offline) and responds to the public key queries by the verifiers in the system.

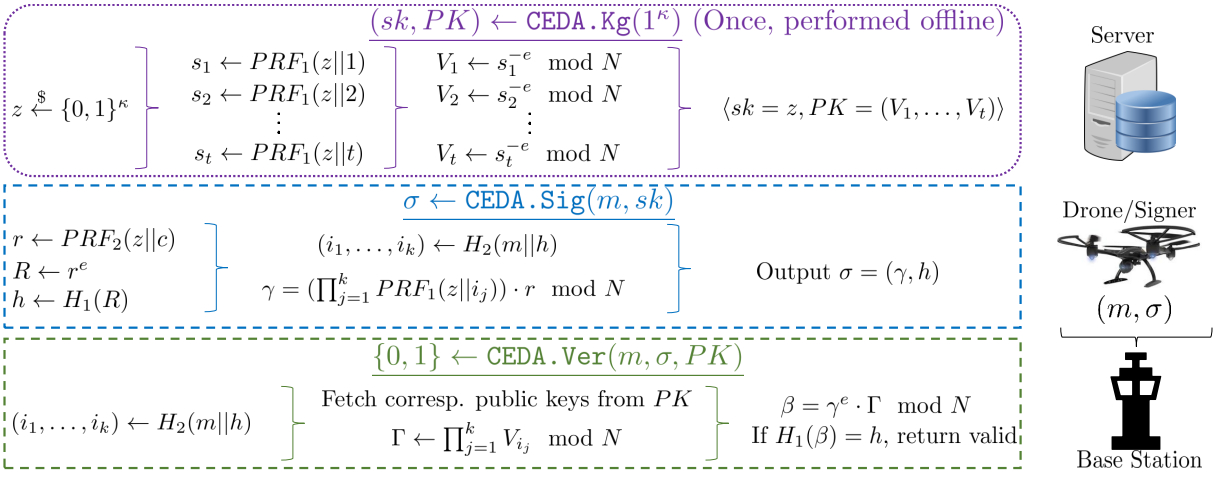

Figure 1: High-level description of *CEDA* algorithms

4 Proposed Scheme

The idea behind the proposed scheme is to leverage the multiplicative property of RSA trapdoor permutation function (Definition 5) to transform one-time HORS [29] signatures into an (practically) unbounded time signature. Specifically, our private key consists of t randomly generated values s_i (that can be deterministically generated with a seed) and the corresponding public key consists of all $V_i \leftarrow (\texttt{Eval}_{\text{RSA}}((N, e), s_i))^{-1} \mod N$ where $i \in \{1, \ldots, t\}$. To sign a message, we compute γ by combining a subset of k selected one-time signature components (i.e., s_i's whose indexes $(i_1, \ldots, i_k)$ are obtained from the message hash output, as in HORS) along with a one-time randomness r to prevent their disclosure. Recall that the release of the private key components with each signature is the main reason that HORS is a one-time signature. We then compute $R \leftarrow \texttt{Eval}_{\text{RSA}}((N, e), r)$ and set the *CEDA* signature as $\sigma = (R, \gamma)$. Upon receiving the signature, the verifier first multiplies the subset of corresponding public keys from PK and calculates Γ. The verifier checks $R = \texttt{Eval}_{\text{RSA}}((N, e), \gamma) \cdot \Gamma$, and returns valid (1) if it holds; otherwise returns invalid (0).

Our scheme consists of the following algorithms.

$(sk, PK) \leftarrow \text{CEDA.Kg}(1^\kappa)$: Given the security parameter κ, this algorithm works as follows:

1. Select *HORS* parameters (t, k) as in Definition 3 and run $\langle (N, e), (N, d) \rangle \leftarrow \texttt{Gen}_{\text{RSA}}(1^\kappa)$ to set $sk' = (N, d)$ and $PK' = (N, e)$ as in Definition 5.
2. Pick $z \xleftarrow{\$} \{0,1\}^\kappa$ and compute $s_i \leftarrow PRF_1(z||i)$ for $i = 1, \ldots, t$.
3. Generate the public keys $V_i \leftarrow \texttt{Eval}_{\text{RSA}}(PK', s_i)$ for $i = 1, \ldots, t$ and set a counter $c \leftarrow 0$.
4. Compute the modular inverse of the public keys $V_i = V_i^{-1} \mod N$ for $i = 1, \ldots, t$.
5. Output the public and private key pair $\langle PK = (V_1, \ldots, V_t), sk = z \rangle$ and the public parameters $params = (PK', t, k)$.

$\sigma \leftarrow \text{CEDA.Sig}(m, sk)$: Given a message $m \in \{0,1\}^*$ to be signed, this algorithm works as follows.

1. Generate $r \leftarrow PRF_2(z||c)$ and $R \leftarrow \texttt{Eval}_{\text{RSA}}(PK', r)$ and increment the counter $c \leftarrow c + 1$.

2. Compute $h \leftarrow H_1(R)$ and $(i_1, \ldots, i_k) \leftarrow H_2(m||h)$ where $\{i_j\}_{j=1}^{k} \in [1, t]$ and $|i_j| = \log_2 t$.

3. Generate $s_{i_j} \leftarrow PRF_1(z||i_j)$ for $j = 1, \ldots, k$ and compute $\gamma = (\prod_{j=1}^{k} s_{i_j}) \cdot r \mod N$ to output the signature as $\sigma = (\gamma, h)$.

$\{0, 1\} \leftarrow$ CEDA.$\mathtt{Ver}(m, \sigma, PK)$: Given a message-signature pair $\langle m, \sigma = (\gamma, h)\rangle$ and $PK = (V_1, \ldots, V_t)$, this algorithm works as follows.

1. Compute $(i_1, \ldots, i_k) \leftarrow H_2(m||h)$ where $\{i_j\}_{j=1}^{k} \in [1, k]$ and $|i_j| = \log_2 t$.

2. Compute $\Gamma \leftarrow \prod_{j=1}^{k} V_{i_j} \mod N$ and $\beta = \mathtt{Eval}_{\mathrm{RSA}}(PK', \gamma) \cdot \Gamma \mod N$.

3. If $H_1(\beta) = h$ holds, output 1 and 0 otherwise.

5 Security Analysis

In the random oracle model [6], we prove that *CEDA* is *EU-CMA* in Theorem 1. In our proof, we ignore terms that are negligible in terms of our security parameters.

Theorem 1. If an adversary $\mathcal{A}$ can break the EU-CMA security of our scheme in time $t_{\mathcal{A}}$ after making q_H hash queries and q_S signature queries, we can build another algorithm $\mathcal{B}$ that runs $\mathcal{A}$ as a subroutine and upon outputting a successful forgery by $\mathcal{A}$, $\mathcal{B}$ can invert the RSA trapdoor one-way permutation function as in Definition 6 in time $t_{\mathcal{B}}$.

$$Adv_{EDA}^{EU\text{-}CMA}(t_{\mathcal{A}}, q_H, q_S) \leq Adv_{RSA}(t_{\mathcal{B}}, q_H, q_S)$$

Proof: Let (N, e) be the output of $\mathtt{Gen}_{\mathrm{RSA}}(1^{\kappa})$ as defined in Definition 5 and $Y = \mathtt{Eval}_{\mathrm{RSA}}((N, e), x)$ be the target challenge value for the algorithm $\mathcal{B}$ on a random input $x \in Z_N^*$. $\mathcal{B}$ takes Y as input and runs as follows.

Algorithm $\mathcal{B}(Y)$:

• *Setup:* $\mathcal{B}$ maintains a list $\mathcal{LM}$, and two tables $\mathcal{HL}_1$ and $\mathcal{HL}_2$ that are all initially empty. $\mathcal{LM}$ stores messages M that are queried to CEDA.$\mathtt{Sig}$ oracle by $\mathcal{A}$. $\mathcal{HL}_1$ and $\mathcal{HL}_2$ store the queries (and responses) to hash functions H_1 and H_2, respectively. $\mathcal{B}$ sets up $RO(.)$ and the simulated public keys to initialize CEDA.$\mathtt{Sig}$ oracle as follows.

- *Setup $RO(.)$ Oracle*: $\mathcal{B}$ implements a function *H-Sim* to handle $RO(.)$ queries to random oracles H_1 and H_2. That is, the cryptographic hash functions H_1 and H_2 are modeled as random oracles via *H-Sim* as follows.

  1) $h_1 \leftarrow$ *H-Sim*$(R, \mathcal{HL}_1)$: If $R \in \mathcal{HL}_1$ then *H-Sim* returns the corresponding value $h_1 \leftarrow \mathcal{HL}_1(R)$. Otherwise, it returns $h_1 \xleftarrow{\$} \{0, 1\}^{l_1}$ as the answer, and inserts (R, h_1) into $\mathcal{HL}_1$.
  2) $h_2 \leftarrow$ *H-Sim*$(M\|h_1, \mathcal{HL}_2)$: If $(M\|h_1) \in \mathcal{HL}_2$ then *H-Sim* returns the corresponding value $h_2 \leftarrow \mathcal{HL}_2(M\|h_1)$. Otherwise, it returns $h_2 \xleftarrow{\$} \{0, 1\}^{l_2}$ as the answer, inserts $(M\|h_1, h_2)$ into $\mathcal{HL}_2$.

- *Setup CEDA.$\mathtt{Sig}$ Oracle*: $\mathcal{B}$ selects parameters (t, k) as in CEDA.Kg Step 1, and creates the simulated *CEDA* public key as follows.

1) $\mathcal{B}$ generates index $j' \xleftarrow{\$} [1,t]$ and sets the challenge public key as $V_{j'} \leftarrow Y$.
2) $\mathcal{B}$ generates $\{s_i \xleftarrow{\$} \{0,1\}^{\mu N}\}_{i=1,i\neq j'}^{t}$ and $\{V_i \leftarrow \texttt{Eval}_{\text{RSA}}((N,e),s_i)\}_{i=1,i\neq j'}^{t}$.
3) $\mathcal{B}$ sets $z \xleftarrow{\$} \{0,1\}^{\kappa}$ and counter $c \leftarrow 0$.
4) Set $sk \leftarrow \{s_i\}_{i=1,i\neq j'}^{t}$, $PK \leftarrow (V_1,\ldots,V_t)$ and public parameters $params \leftarrow (t,k,N,e,c)$.

• *Execute* $(M^*,\sigma^*) \leftarrow \mathcal{A}^{RO(.),CEDA.Sig_{sk}(\cdot)}(PK)$: $\mathcal{B}$ handles $\mathcal{A}$'s queries as follows:

- Queries of $\mathcal{A}$: $\mathcal{A}$ can query $RO(.)$ and CEDA.$\texttt{Sig}(\cdot)$ oracles on any message of its choice up to q_H and q_S times, respectively.

1) *Handle $RO(.)$ queries*: $\mathcal{A}$'s queries on H_1 and H_2 are handled by H-Sim function as described above.
2) *Handle CEDA.$\texttt{Sig}$ queries*: To answer $\mathcal{A}$'s signature queries CEDA.$\texttt{Sig}(\cdot)$ on any message of its choice M, $\mathcal{B}$ inserts M into $\mathcal{LM}$ and continues as follows.
   i) Pick $r \in \mathbb{Z}_N^*$ and compute $R' \leftarrow \texttt{Eval}_{\text{RSA}}((N,e), r)$.
   ii) $i_j \xleftarrow{\$} [1,\ldots,t], j = 1,\ldots,k$.
   iii) $R \leftarrow R' \cdot \prod_{j=1}^{k} V_{i_j}^{-1} \mod N$.
   iv) $h \xleftarrow{\$} \{0,1\}^{l_1}$ and insert (R,h) in $\mathcal{HL}_1$.
   v) If $(H(M||h)) \in \mathcal{HL}_2$, $\mathcal{B}$ aborts. We call this event BAD1. Else, it inserts $(H(M\|h), \langle i_1 \ldots, i_k\rangle)$ in $\mathcal{HL}_2$.
   vi) Set $\sigma = (\gamma, h)$ where $\gamma = r$, and return σ to $\mathcal{A}$.

- Forgery of $\mathcal{A}$: Finally, $\mathcal{A}$ outputs a forgery for PK as (M^*,σ^*), where $\sigma^* = (\gamma^*, h^*)$. By Definition 2, $\mathcal{A}$ wins the EU-CMA experiment for $CEDA$ if the below conditions hold.

   i) CEDA.$\texttt{Ver}(M^*,\sigma^*,PK) = 1$
   ii) $M^* \notin \mathcal{LM}$

• *$\mathcal{B}$'s Attempt to Invert RSA Trapdoor Permutation*: If $\mathcal{A}$ fails in the EU-CMA experiment for $CEDA$, $\mathcal{B}$ also fails in inverting the RSA trapdoor permutation function as in Definition 5, and therefore, $\mathcal{B}$ *aborts* and returns 0.

Otherwise, if $\mathcal{A}$ outputs a successful forgery (M^*,σ^*), by behaving similar to $\mathsf{F}_A(x)$, as in [7, Lemma 1], $\mathcal{B}$ can rewind $\mathcal{A}$ to get a second forgery $(M^*, \tilde{\sigma} = \langle\tilde{\gamma},\tilde{h}\rangle)$ where $\gamma^* \neq \tilde{\gamma}$ and $h^* = \tilde{h}$ with an overwhelming probability. Given $CEDA$ forgeries $(M^*, \sigma^* = \langle\gamma^*, h^*\rangle)$ and $(M^*, \tilde{\sigma} = \langle\tilde{\gamma},\tilde{h}\rangle)$ on PK where $(M^*||h^*) = (M^*||\tilde{h})$, based on [7, Lemma 1], we know that $H(M^*\|h^*) \neq H(M^*\|\tilde{h})$. Then $\mathcal{B}$ can attempt to break RSA trapdoor permutation function if either of the following conditions holds.

- If $(M^*\|h^* \in \mathcal{HL}_2)$ and $(j' \in (i_1^* \ldots, i_k^*))$ then $(j' \notin (\tilde{i}_1 \ldots, \tilde{i}_k))$, where $(i_1^* \ldots, i_k^*) \leftarrow \mathcal{HL}_2(M^*\|h^*)$ and $(\tilde{i}_1 \ldots, \tilde{i}_k) \leftarrow \mathcal{HL}_2(M^*\|\tilde{h})$. We recall this event as GOOD_1.

- If $(M^*\|h^* \in \mathcal{HL}_2)$ and $(j' \notin (i_1^* \ldots, i_k^*))$ then $(j' \in (\tilde{i}_1 \ldots, \tilde{i}_k))$, where $(i_1^* \ldots, i_k^*) \leftarrow \mathcal{HL}_2(M^*\|h^*)$ and $(\tilde{i}_1 \ldots, \tilde{i}_k) \leftarrow \mathcal{HL}_2(M^*\|\tilde{h})$. We recall this event as GOOD_2.

In a case that none of the above conditions holds, $\mathcal{B}$ aborts and fails to break RSA, otherwise, it works as follows.

- Case 1: If $j' \in (i_1^* \ldots, i_k^*)$ and $j' \notin (\tilde{i}_1 \ldots, \tilde{i}_k)$, set $\widetilde{x} \leftarrow \gamma^* \prod_{j=1}^{k} s_{\tilde{i}_j} / \tilde{\gamma} \prod_{j=1, j\neq j'}^{k} s_{i_j^*} \bmod N$.
- Case 2: If $j' \notin (i_1^* \ldots, i_k^*)$ and $j' \in (\tilde{i}_1 \ldots, \tilde{i}_k)$, set $\widetilde{x} \leftarrow \tilde{\gamma} \prod_{j=1}^{k} s_{i_j^*} / \gamma^* \prod_{j=1, j\neq j'}^{k} s_{\tilde{i}_j} \bmod N$.

Then, if $\widetilde{x} = x$ implies that $\mathcal{B}$ has inverted RSA permutation function without any knowledge of the trapdoor.

- Success Probability Analysis: We analyze the events that are needed for $\mathcal{B}$ to successfully invert RSA as follows.
- BAD1: $\mathcal{B}$ may abort in the simulation phase when the adversary queries the CEDA.`Sig` oracle. This event happens when the randomly drawn $(i_1, \ldots, i_k)$ already exists in $\mathcal{HL}_2$. This can happen with the probability $(q_H-1)q_S/2^l$.
- ACC: The success probability of $\mathcal{A}$ to win the game in Definition 2 is as in [7, Lemma 1].
- FRK: $\mathcal{B}$ receives two valid forgeries from $\mathcal{A}$ for the target message.
- $\overline{\text{BAD2}}$: If $\mathcal{A}$ successfully outputs a forgery in each of the runs, then $\mathcal{B}$ will break RSA if ($\text{GOOD}_1 \vee \text{GOOD}_2$) happens for the forged signatures. ($\text{GOOD}_1 \vee \text{GOOD}_2$) can happen with a non-negligible probability of $2k(t-k)/t^2$. Note that given the random behavior of our hash function, we consider the probability $((M^*||h^* \notin \mathcal{HL}_2) \vee (M^*\|\tilde{h} \notin \mathcal{HL}_2))$ to be negligible in the case of the above event.

We bound the success probability of $\mathcal{A}$ as defined in [7, Lemma 1] as $\text{ACC} \geq \epsilon_{\mathcal{A}} - \Pr[\text{BAD1}]$. The probability that BAD1 occurs can be upper-bounded by $(q_H-1)q_S/2^{l_2}$, and therefore, $\text{ACC} \geq \epsilon_{\mathcal{A}} - (q_H-1)q_S/2^{l_2}$.

The probability of $\mathcal{B}$ in breaking RSA is given by:

$$\begin{aligned}
\epsilon_{\mathcal{B}} &\geq \text{FRK} \cdot \overline{\text{BAD2}} \\
&\geq \left(\frac{\text{ACC}^2}{q_H + q_S} - \frac{1}{2^{l_2}} \right) \cdot \overline{\text{BAD2}} \\
&\geq \left(\frac{\epsilon_{\mathcal{A}}^2}{(q_H + q_S)} - \frac{2((q_H-1)q_S)}{2^{l_2}(q_H + q_S)} - \frac{1}{2^{l_2}} \right) \cdot \frac{2k(t-k)}{t^2}
\end{aligned}$$

Target Collision of Hash Function (H_2): Following the work of [28], the security of our scheme relies on the subset resiliency of the underlying hash function. We "salt" the hash of each signature with a one-time randomness so that $\mathcal{A}$ does not know the internal state of the hash function when they want to compute a collision. Therefore, we reduce the hardness of our signature scheme from collision resistance to target-collision resistance. Therefore, considering k subsets in the hash output, and the number of $\mathcal{A}$'s signature queries q_S, the target collision resiliency of our hash function is $q_S \cdot k!/2^{l_2}$. The $k!$ factor comes in place since we should consider different permutations of the indexes in the hash output which would potentially result in a collision and forgery.

6 Performance Analysis and Comparison

We first compare the analytical costs of *CEDA* with its counterparts and then describe our evaluation metrics along with the experimental setup. We then present our detailed experimental results on both commodity hardware and an ARM processor. Note that we only compare our scheme with the state-of-the-art digital signatures with a constant-size key/token storage overhead. Moreover, we also consider

Table 3: Analytical Performance Comparison of $CEDA$ and its Counterparts

Scheme	Signer		Transmission	Verifier	
	Private Key†	***Signature Generation***¶	***Signature***†	***Public Key***†	***Signature Verification***¶
RSA	$\|N\| + \|d\|$	Exp_d	$\|N\|$	$\|N\| + \|e\|$	Exp_e
ECDSA	$\|q'\|$	$Emul + H + Mul_{q'}$	$\|q'\| + \|H\|$	$\|q'\|$	$1.3 \cdot Emul + Eadd + H$
BPV-ECDSA	$\|q'\| + T1$	$v \cdot Eadd + H + Mul_{q'}$	$\|q'\| + \|H\|$	$\|q'\|$	$1.3 \cdot Emul + Eadd + H$
Ed25519	$\|q'\|$	$Emul_{25519}+$ $2H + Mul_{q'}$	$\|q'\| + \|H\|$	$\|q'\|$	$1.3 \cdot Emul_{25519}+$ $Eadd_{25519} + H$
SCRA-C-RSA	$\|N\| + T2$	$L \cdot Mul_N$	$\|N\| + \|H\| + \kappa$	$\|N\| + \|e\| + \kappa$	$Exp_e + L \cdot H + L \cdot Mul_N$
$CEDA$	$\|z\| + \|N\|$	$(k+3) \cdot H + Exp_e$	$\|N\| + \|H\|$	$\|N\| + \|e\| + PK$	$Exp_e + k \cdot Mul_N$

¶ Exp_e and Exp_d denote exponentiation over the small exponent e and large exponent d, respectively. $Emul$ and $Eadd$ denote the costs of EC scalar multiplication over modulus p', and EC addition over modulus p, respectively. $Emul$ and $Eadd$ are performed in secp256r1, where $Emul_{25519}$ and $Eadd_{25519}$ are performed on twisted Edwards' curve. H and $Mul_{q'}$ denote a cryptographic hash and a modular multiplication over modulus q', respectively. We omit the constant number of negligible operations if there is an expensive operation (e.g., integer additions are omitted if there is an $Emul$ or Exp_e). We use double-point scalar multiplication for verifications of ECC based schemes ($1.3 \cdot Emul$ instead of $2 \cdot Emul$ [17]).
Suggested parameters for v, L, k are 32 [11], 32 [37], and 26, respectively.
† For $\kappa = 128$, the parameter sizes are: $|N| = 3072$ bit, $|e| = 17$ bit, $|d| \approx 3072$ bit, and $|z| = 128$ bit. The size of the pre-computation tables with the suggested parameters for BPV-ECDSA [11], SCRA-C-RSA [37] and $CEDA$ are 384 KB, 10KB [11] and 2MB [37] for PK, $T1$ and $T2$, respectively. For ECC-based schemes, (p', q') are ECC parameters where $|p'| = |q'| = 256$ bit.

optimization techniques such as constant storage pre-computation [11] and efficient curves [10]. Further note that in [37], authors proposed three instantiations of SCRA: (i) SCRA-C-RSA (ii) SCRA-BGLS (iii) SCRA-NTRU. We compare the cost of $CEDA$ with SCRA-C-RSA since it achieves the lowest end-to-end delay among these three schemes with a mid-size table stored at the signer's side [37].

6.1 Analytical Performance Comparison

Table 3 shows the analytical comparison of $CEDA$ with its state-of-the-art counterparts.

Signer Computation and Storage: In $CEDA$, signature generation only requires an exponentiation over the small exponent e and a small-constant number of hash calls, which have an (almost) negligible overhead (implemented with highly optimized Blake2 [5]). The small exponent is selected as $e = 65537$ to ensure the security, while enabling the computational efficiency as such an exponentiation can only be done with 16 squarings and a single multiplication via square-and-multiply algorithm. Moreover, $CEDA$ has a much smaller private key size than that of its delay-aware variants as well as the RSA signature, since the signer does not store a pre-computed table or the RSA private key d.

RSA and ECDSA require an exponentiation over large exponent and elliptic curve scalar multiplication(s), respectively, both of which are considered as expensive computations. BPV-ECDSA eliminates the scalar multiplication in ECDSA [4] in exchange of some elliptic curve additions [11]. However, it requires storing a pre-computation table at the signer's side. Ed25519 scheme [10] uses efficient twisted Edwards' curve to perform scalar multiplications. It also has a very compact private key. SCRA-C-RSA [37] only requires L multiplications to compute the signature, where L is suggested to be 32. However, this scheme requires a very large private key of 2MB, which may not be feasible for some resource-constrained devices.

Signature Transmission: $CEDA$ has a compact signature that has the same size with standard RSA signature scheme. However, elliptic curve based schemes offer more compact signatures. More specif-

ically, signature length in RSA-based schemes, including *CEDA*, require at least $|N| + |H|$ bits where $|N| = 3072$ bits for $\kappa = 128$ bit security. On the other hand elliptic curve based schemes require a signature size of $|q'| + |H|$ where $|q'| = 256$ bits.

Verifier Computation and Storage: *CEDA* has an ultra efficient verification algorithm since it only requires an exponentiation over e and k multiplications, where k is suggested to be 26. However, *CEDA* has a relatively large public key size, that requires storing a table. This table has a size of $t \cdot |N|$, where $t = 1024$ and $|N| = 3072$ bits. On the other hand, all elliptic curve based counterparts have a very small public key of size 32 bytes, but they require a double scalar multiplication for verification. Double scalar multiplication can be accelerated with Shamir's trick [17], however, this is still a very expensive operation, and to the best of our knowledge, there are no pre-computation methods to speed-up this operation. RSA verification is the fastest among all schemes, since it only requires an exponentiation over e. It also has a compact public key size of $|N| + |e|$. SCRA-C-RSA requires exponentiation over e along with L hash and multiplication calls, where L is suggested to be 32 [37]. As for the public key size, it only requires and additional κ bits to be stored, in addition to traditional RSA [30].

Our analytical analysis shows that *CEDA* only requires a small-constant number of inexpensive operations at the signer's and verifier's sides, which makes it a suitable alternative for delay-aware applications. It has a compact private key and signature size as compared to that of its delay-aware signature alternatives. However, it can be seen that elliptic curve-based counterparts offer more compact private key and signatures than *CEDA*, but with the cost of a large end-to-end delay. The main limitation of *CEDA* is its relatively large public key size, which can be readily stored by verifiers for many real-life applications.

6.2 Experimental Evaluation

Evaluation Metrics: We implemented *CEDA* both on an IoT device (ARM Cortex A53) and commodity hardware. We also ran our counterparts on both devices to compare the signature generation and verification times. Moreover, we discuss the signer's and verifier's storage, along with the transmission requirement of each signature scheme.

Software Libraries and Implementation: We developed two implementations of *CEDA* in C, one with MIRACL [32] and the other with GMP [15]. We observed that GMP implementation is significantly faster, and therefore we present our results in GMP. We use Blake2 as our cryptographic hash function and PRF due to its high efficiency [5]. We use portable implementation of Blake2 hash, b2 library. We have open-sourced our implementation of *CEDA* for wide adaptation and comparison.

`https://github.com/ozgurozmen/CEDA`

Aside from the hash functions and RSA parameters, the security of our scheme relies on the selection of parameters (t, k). The security of these parameters depend on the number of k-out-of-t different combinations and also the target collision $(k!/2^{l_2})$ as described in Section 5. We selected $t = 1024$ and $k = 26$ which guarantees 2^{172} different combinations and a target collision probability of $1/2^{171}$. Different t and k parameters can also be selected to instantiate *CEDA*, which offer a trade-off between computation and storage. For instance, $(t = 256, k = 32)$ also provide a high security level, with smaller storage but slower computation. Since $|N| = 3072$ provides approximately $\kappa = 128$-bit security, all in all, our current *CEDA* implementation offers $\kappa = 128$-bit security.

Table 4: Experimental Performance Comparison of *CEDA* and its Counterparts on an Intel Processor

Scheme	Signer		Transmission	Verifier		End-to-End Delay (μs)
	Signature Generation (μs)	Private Key (Byte)	Signature Size (Byte)	Signature Verification (μs)	Public Key (Byte)	
RSA	8083.26	768	416	47.74	386	8131.00
ECDSA	725.38	32	64	927.30	32	1652.68
BPV-ECDSA	149.60	10272	64	927.30	32	1076.9
Ed25519	132.61	32	64	335.95	32	468.56
SCRA-C-RSA	88.67	2000000	432	164.85	384	253.52
CEDA	**55.33**	416	416	**115.45**	393600	**170.78**

We benchmarked the ECDSA implementation in MIRACL library [32]. We applied BPV pre-computation technique [11] to ECDSA implementation of MIRACL. For Ed25519, we used the Supercop implementation [10]. Note that BPV pre-computation technique cannot be directly incorporated into Ed25519 scheme, since the randomness is generated deterministically with the message that is being signed. We also benchmarked RSA [30] with GMP library in C [15]. SCRA-C-RSA was implemented in MIRACL library in [37], however, our experiments showed us that MIRACL is significantly slower than GMP for modular exponentiations and multiplications. Therefore, for the purpose of fairness, we measured SCRA-C-RSA costs with GMP library. Moreover, we observed that authors selected the small exponent in RSA as $e = 3$, that is not recommended [14]. Therefore, we calculated SCRA-C-RSA costs with $e = 65537$ (as in *CEDA* implementation).

Hardware Configurations: We benchmarked our scheme and its counterparts on an ARM Cortex A53 processor as the IoT device. ARM Cortex A53 is a low-cost and low energy consuming (can work with small batteries) device with a powerful processor [34]. Therefore, it is highly preferred in IoT applications [27]. We used a laptop equipped with Intel Core i7 6700HQ 2.6 GHz processor and 12GB RAM as the commodity hardware.

6.3 Performance Evaluation

Tables 1 and 4 depict the experimental results of *CEDA* and its counterparts on ARM Cortex A53 and commodity hardware, respectively.

IoT Device: Our experiments on ARM Cortex A53 show that *CEDA* is the fastest signature scheme among its counterparts. *CEDA* outperforms all its counterparts in terms of signature generation and verification speeds (the only exception is RSA verification, however the signature generation of RSA is very expensive). More specifically, *CEDA* has 1.56×, 1.98×, and 3.03× lower end-to-end delay as compared to SCRA-C-RSA, Ed25519, and BPV-ECDSA (as its most efficient counterparts), respectively. Although *CEDA* requires a larger storage requirement at the verifier's side, due to the larger public key ($\approx$ 393 KB), it is still highly achiveable with the storage capabilities of IoT devices such as ARM Cortex A53.

Energy consumption hinders the full adoption of cryptographic protocols to IoT systems. Therefore, it is highly useful to provide an energy-efficient cryptographic protocol for IoT systems. Note that computational energy consumption can be calculated with the formula $E = V \cdot I \cdot t$, where V is the voltage processor is taking, I is the current drawn by the processor and t is the computation time. Considering most IoT processors work with constant currents and voltages in active mode, computation time should be optimized to decrease the energy consumption. Thus, computational efficiency of

$CEDA$ drastically reduces the energy consumption and we believe that it is the most suitable signature scheme to be deployed in energy-critical applications.

Commodity Hardware: The signature generation of $CEDA$ is 1.60× faster than that of SCRA-C-RSA (the fastest counterpart), which has a very large private key (2MB). We note that $CEDA$ can generate *18,070 signatures per second*, which can meet the high throughput signing requirements of various real-life applications. For instance, as discussed in Section 1.2, vehicular networks may require a significantly large throughput for signature generation [2]. With the hardware configuration described, $CEDA$ offers a signing speed way above this requirement, which can be suitable for infrastructure-to-vehicle communication. Therefore, we believe $CEDA$ can potentially meet the needs of even the most stringent requirements of high signing throughput applications.

$CEDA$ signature verification is also 1.43× and 2.91× faster than that of SCRA-C-RSA and Ed25519 (the fastest counterpart with reasonable end-to-end delay), respectively. Note that standard RSA has 2× faster verification than $CEDA$. However, its signature generation is 146.17× slower than $CEDA$, which is not suitable for delay-aware applications. The signature verification time is highly critical for applications that require a fast response to the commands/messages. We believe that $CEDA$ is highly suitable for such applications with a very fast verification and end-to-end delay. Specifically, *verification throughput of $CEDA$ is 8,660 signatures per second.* However, as depicted in Table 4, $CEDA$ requires storing a public key of size almost 393 KB at the verifier's side, when $t = 1024$. Therefore, if the verifier is storage-limited, different parameters (e.g. $t = 256$, $k = 32$) can be used to instantiate $CEDA$ with a storage-computation trade-off.

7 Conclusion

In this paper, towards addressing the authentication requirements of time-critical applications, we created a novel delay-aware digital signature scheme that we refer to as *Compact Energy and Delay-aware Authentication* ($CEDA$). $CEDA$ achieves the lowest end-to-end cryptographic delay among all of its counterparts by offering the fastest signature generation along with a highly efficient verification. Moreover $CEDA$ requires only a small-constant size private key and signature, which are smaller than its most efficient delay-aware counterparts. Our experiments on ARM and Intel processors further confirmed the significant speed advantages of $CEDA$ over its counterparts with compact signer storage overhead. On the other hand, $CEDA$ has a larger public key size than that of its counterparts. Overall, by offering the lowest end-to-end delay with small private key and signature sizes, $CEDA$ is an ideal authentication tool for delay-aware critical systems such as energy delivery (e.g., smart-grids) and mobile cyber-physical systems (e.g., vehicular and aerial drone networks). We open-sourced our implementation of $CEDA$ for public testing and adaptation purposes.

References

[1] Ieee standard communication delivery time performance requirements for electric power substation automation. *IEEE Std 1646-2004*, pages 1–24, 2005.

[2] Ieee guide for wireless access in vehicular environments (wave) - architecture. *IEEE Std 1609.0-2013*, pages 1–78, March 2014.

[3] D. Aggarwal and U. Maurer. Breaking rsa generically is equivalent to factoring. *IEEE Transactions on Information Theory*, 62(11):6251–6259, Nov 2016.

[4] American Bankers Association. *ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 1999.

[5] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. Sha-3 proposal blake. Submission to NIST (Round 3), 2010.

[6] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and Communications Security (CCS '93)*, pages 62–73, NY, USA, 1993. ACM.

[7] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 390–399, New York, NY, USA, 2006. ACM.

[8] Piotr Berman, Marek Karpinski, and Yakov Nekrich. Optimal trade-off for merkle tree traversal. *Theor. Comput. Sci.*, 372(1):26–36, March 2007.

[9] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical stateless hash-based signatures. In *Advances in Cryptology – EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 368–397. Springer Berlin Heidelberg, April 2015.

[10] DanielJ. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[11] Victor Boyko, Marcus Peinado, and Ramarathnam Venkatesan. Speeding up discrete log and factoring based schemes via precomputations. In *Advances in Cryptology — EUROCRYPT'98: International Conference on the Theory and Application of Cryptographic Techniques Espoo, Finland, May 31 – June 4, 1998 Proceedings*, pages 221–235, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[12] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. Xmss - a practical forward secure signature scheme based on minimal security assumptions. In *Proceedings of the 4th International Conference on Post-Quantum Cryptography*, PQCrypto'11, pages 117–129, Berlin, Heidelberg, 2011. Springer-Verlag.

[13] D. Catalano, M. D. Raimondo, D. Fiore, and R. Gennaro. Off-line/on-line signatures: Theoretical aspects and experimental results. Public Key Cryptography (PKC), pages 101–120. Springer-Verlag, 2008.

[14] Don Coppersmith. Small solutions to polynomial equations, and low exponent rsa vulnerabilities. *Journal of Cryptology*, 10(4):233–260, Sep 1997.

[15] Torbjörn Granlund et al. GNU multiple precision arithmetic library 6.1.2. `https://gmplib.org/`.

[16] S. Even, O. Goldreich, and S. Micali. Online/offline digital signatures. In *Proceedings on Advances in Cryptology (CRYPTO '89)*, pages 263–275. Springer-Verlag, 1989.

[17] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[18] John Harding, Gregory Powell, Rebecca Yoon, Joshua Fikentscher, Charlene Doyle, Dana Sade, Mike Lukuc, Jim Simons, and Jing Wang. Vehicle-to-Vehicle Communications: Readiness of V2V Technology for Application. *U.S. Department of Transportation National Highway Traffic Safety Administration (NHTSA)*, August 2014.

[19] Andreas Hülsing, Joost Rijneveld, and Peter Schwabe. Armed SPHINCS - computing a 41 KB signature in 16 KB of RAM. In *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography*, pages 446–470, March 2016.

[20] Q. Li and G. Cao. Multicast authentication in the smart grid with one-time signature. *IEEE Transactions on Smart Grid*, 2(4):686 –696, December 2011.

[21] M. Luk, A. Perrig, and B. Whillock. Seven cardinal properties of sensor network broadcast authentication. In *Proceedings of 4th ACM workshop on security of ad hoc and sensor networks*, SASN '06, pages 147–156, New York, NY, USA, 2006. ACM.

[22] A. Lysyanskaya, R. Tamassia, and N. Triandopoulos. Multicast authentication in fully adversarial networks. In *IEEE Symposium on Security and Privacy*, pages 241 –253, May 2004.

[23] S. S. Manvi, M. S. Kakkasageri, and D. G. Adiga. Message authentication in vehicular ad hoc networks: Ecdsa based approach. In *Proceedings of the 2009 International Conference on Future Computer and Communication*, ICFCC '09, pages 16–20, Washington, DC, USA, 2009. IEEE Computer Society.

[24] Ralph C. Merkle. A certified digital signature. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 218–238, New York, NY, USA, 1989. Springer-Verlag.

[25] D. Naccache, D. M'Raïhi, S. Vaudenay, and D. Raphaeli. Can D.S.A. be improved? Complexity trade-offs with the digital signature standard. In *Proceedings of the 13th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '94)*, pages 77–85, 1994.

[26] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and secure source authentication for multicast. In *Proceedings of Network and Distributed System Security Symposium*, February 2001.

[27] A. Raza, A. A. Ikram, A. Amin, and A. J. Ikram. A review of low cost and power efficient development boards for iot applications. In *2016 Future Technologies Conference (FTC)*, pages 786–790, Dec 2016.

[28] L. Reyzin and N. Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *Proceedings of the 7th Australian Conference on Information Security and Privacy (ACIPS '02)*, pages 144–153. Springer-Verlag, 2002.

[29] Leonid Reyzin and Natan Reyzin. Better than biba: Short one-time signatures with fast signing and verifying. In Lynn Batten and Jennifer Seberry, editors, *Information Security and Privacy: 7th Australasian Conference, ACISP 2002 Melbourne, Australia, July 3–5, 2002 Proceedings*, pages 144–153, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[30] R.L. Rivest, A. Shamir, and L.A. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[31] A. Shamir and Y. Tauman. Improved online/offline signature schemes. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 355–367, London, UK, 2001. Springer-Verlag.

[32] Shamus. Multiprecision integer and rational arithmetic c/c++ library (MIRACL). `http://www.certivox.com/miracl/miracl-download/`. Last Accessed on 09/02/2014.

[33] T. Tesfay and J. Y. Le Boudec. Experimental comparison of multicast authentication for wide area monitoring systems. *IEEE Transactions on Smart Grid*, PP(99), 2017.

[34] Vladimir Vujović and Mirjana Maksimović. Raspberry pi as a sensor web node for home automation. *Comput. Electr. Eng.*, 44(C):153–171, May 2015.

[35] Q. Wang, H. Khurana, Y. Huang, and K. Nahrstedt. Time valid one-time signature for time-critical multicast data authentication. In *INFOCOM 2009, IEEE*, April 2009.

[36] A. A. Yavuz. An efficient real-time broadcast authentication scheme for command and control messages. *IEEE Transactions on Information Forensics and Security*, 9(10):1733–1742, Oct 2014.

[37] A. A. Yavuz, A. Mudgerikar, A. Singla, I. Papapanagiotou, and E. Bertino. Real-time digital signatures for time-critical networks. *IEEE Transactions on Information Forensics and Security*, 12(11):2627–2639, Nov 2017.

[38] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1):22–32, Feb 2014.

[39] Chenxi Zhang, Pin-Han Ho, and Janos Tapolcai. On batch verification with group testing for vehicular communications. *Wireless Networking*, 17(8):1851–1865, November 2011.