CSProp: Ciphertext and Signature Propagation Low-Overhead Public-Key Cryptosystem for IoT Environments

Fatemah Alharbi, *Taibah University, Yanbu;* Arwa Alrawais, *Prince Sattam Bin Abdulaziz University;* Abdulrahman Bin Rabiah, *University of California, Riverside, and King Saud University;* Silas Richelson and Nael Abu-Ghazaleh, *University of California, Riverside*

https://www.usenix.org/conference/usenixsecurity21/presentation/alharbi

This paper is included in the Proceedings of the 30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the 30th USENIX Security Symposium is sponsored by USENIX.

CSProp: Ciphertext and Signature Propagation Low-Overhead Public-Key Cryptosystem for IoT Environments

Fatemah Alharbi
Taibah University, Yanbu

Arwa Alrawais
Prince Sattam Bin Abdulaziz University

Abdulrahman Bin Rabiah
University of California, Riverside
King Saud University

Silas Richelson
University of California, Riverside

Nael Abu-Ghazaleh
University of California, Riverside

Abstract

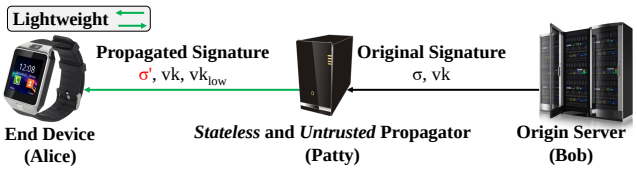
Cryptographic operations can be prohibitively expensive for IoT and other resource-constrained devices. We introduce a new cryptographic primitive which we call *Ciphertext and Signature Propagation* (CSProp) in order to deliver security to the weak end-devices. CSProp is a cryptographic propagation algorithm whereby an untrusted machine sitting upstream of a lightweight device can modify an authenticated message so it can be efficiently verified. Unlike proxy-based solutions, this upstream machine is stateless and untrusted (making it possible for any device to serve that role), and the propagated signature is mathematically guaranteed to be valid only if the original signature is also valid. CSProp relies on RSA security and can be used to optimize any operations using the public key such as signature validation and encryption, which our experiments show are the most common public key operations in IoT settings. We test CSProp by using it to extend DNSSEC to edge devices (validation), and to optimize the performance of TLS (validation and encryption) on a range of resource constrained devices. CSProp reduces DNSSEC validation latency by 78x and energy consumption by 47x on the Raspberry Pi Zero. It reduces TLS handshake latency and energy by an average of 8x each. On an Arduino-based IoT board, CSProp significantly outperforms traditional RSA public key operations (*e.g.,* 57x and 36x reductions in latency and energy consumption, respectively, for encryption).

1 Introduction and Roadmap

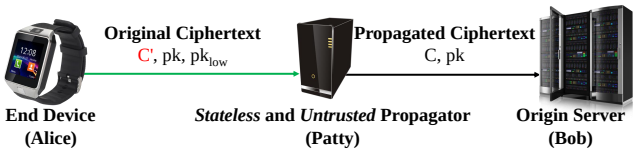
Critical infrastructure on the Internet relies on the distribution of roles and responsibilities over several nodes. The interaction between nodes often occurs over secure channels to provide the required level and type of security (*i.e.,* confidentiality, integrity, availability − the CIA triad). Operating securely in constrained environments is one of the primary challenges facing the wide-scale deployment of Internet of Things (IoT) and other embedded systems on the edge of the Internet. The problem is that the cryptographic algorithms used to secure interactions between well-provisioned desktop and server environments are computationally prohibitive for resource-poor, battery operated devices. By the year 2025, it is estimated that the number of IoT devices will be over 75 billion [41]; thus, it is essential to develop security solutions for them.

We focus on a security problem which arises when resource-constrained devices are added to a secure network of more capable machines. If the security protocols/primitives used by the network are too computationally intensive for the small device, then either (1) performance will suffer if we attempt to use the primitives as is; (2) security will suffer, for example, if we relegate participation to a resource rich gateway or proxy; or (3) security for the network must be overhauled so the new device can participate. In standardized large-scale networks, (3) is likely not an option due to the large development time, and the lack of backward compatibility, and so (2) will be chosen to avoid the performance and functionality cost.

In this paper, we contribute a new cryptographic primitive we call *Ciphertext and Signature Propagation* (CSProp). When used for signature propagation, CSProp allows a capable machine (Patty in Figure 1a), even one that is *stateless and untrusted (*e.g., *a certificate is not required to authenticate it)* sitting upstream of a lightweight device to bear the majority of the cost of verification. Specifically, Patty modifies and forwards (propagate) an authenticated message so it can be efficiently verified by a lesser machine. Importantly, it is cryptographically guaranteed that the propagated signature verifies correctly only if the original signature does. The trivial solution where Patty simply forwards Bob's (data, signature) pair directly to Alice puts unacceptable strain on Alice's resources. Another trivial solution where Patty simply verifies Bob's signature herself and forwards only the data to Alice is undesirable from a security point of view as it requires Alice to be trusted, and also opens the door for an attacker who targets the link between Alice and Patty. Likewise, when used for ciphertext propagation, CSProp allows Patty (see Figure 1b) to perform the majority of the computational overhead caused by public key encryption. More precisely, Alice partially en-



(a) Signature Propagation



(b) Ciphertext Propagation

Figure 1: Figure 1: High Level Overview of CSProp

crypts the message and forwards a lightweight ciphertext to Patty. Patty completes the encryption operation performing the more expensive portion of the operation. The construction of CSProp (see Section 3) guarantees the security of the original message assuming only that the standard public key encryption (e.g., RSA) is secure.

CSProp differs from a small number of prior proposals that use a proxy [28, 52] to reduce the cost of encryption in two important ways: (1) We do not require the proxy to be trusted since the security is obtained by construction; and (2) CSProp is backwards compatible with RSA, making it straightforward to deploy. Specifically, the construction provides security guarantees that there is no way for Patty to produce a valid lightweight propagated signature, except by propagating an original valid signature from Bob. Thus, CSProp securely implements a lightweight channel between Alice and Bob, without requiring any modifications to the protocol at Bob (i.e., providing backward compatibility at the server). CSProp requires no state, making it possible to change the role of Patty, even at the granularity of each cryptographic operation. We provide related background and preliminaries in Section 2 and present a formal definition of the new primitive, as well as an instantiation based on RSA in Section 3.

CSProp can optimize public-key operations which include signature verification and encryption, but not operations that use the private key such as signing and decryption. Public-key operations are typically executed at the client's end specially when using Internet protocols such as the Domain Name System SECurity extension (DNSSEC) and the Transport Layer Security (TLS) protocols, and when generating data that is being forwarded to an upstream server. We conduct a measurement study of the traffic generated by an IoT camera, discovering that TLS signature verification and encryption operations account for the majority of the Public Key cryptographic operations.

We apply CSProp to improve the performance of two security protocols on IoT devices in Section 4: DNSSEC [39], a secure extension of the Domain Naming System protocol, and the Transport Layer Security (TLS) [83] protocol which is the

Table 1: Glossary

Acronym	Definition	Acronym	Definition
sk	Secret key	pk	Public key
pk_{low}	Low public key	vk	Verification key
vk_{low}	Low verification key	N	Public modulus
e	Public exponent	e_{low}	Low public exponent
d	Private exponent	M	Plaintext message
C	Ciphertext	C'	Partial decrypted ciphertext
h	Message digest	σ	Digital signature
σ′	Partial verified digital signature	K	Pre-master key
H	Hash function	$\mathcal{A}$	Adversary
$\mathcal{C}$	Challenger	$\mathcal{P}$	Computational problem
ϕ	Totient function	A	Address record
DS	Delegation signer record	DNSKEY	DNS Key record
RRset	A set of DNS records of same type	RRSIG	DNSSEC signature
KSK	Zone's key signing key	ZSK	Zone's zone signing key
RRsetA	RRset of A record(s) type	RRsetDS	RRset of DS record(s) type
RRsetDNSKEY	RRset of DNSKEY records type	$\overline{\mathsf{M}}$	A padded version of M

backbone of secure communication on the Internet. DNSSEC requires a sequence of signature validations (public-key operations) to validate a DNS response through the sequence of DNS servers that are used to obtain it. TLS also requires signature validation as part of connection establishment to authenticate the ends of the connection, but also uses encryption to establish a session key, both of which are operations that use the public key. In an IoT setting, often such operations are offloaded to a third server (*e.g.,* a DNS resolver or a default gateway), thus shielding the end devices from overhead of encryption and verification. However, the last hop is left unprotected: for example, a recent attack [9] has shown that DNS cache poisoning can be performed between the end device and the resolver to directly poison the OS-wide DNS cache of the victim's system. CSProp can be used to secure the end devices by having the resolver propagate the signatures forward for efficient verification.

In Section 5, we evaluate the impact of CSProp using experiments on three generations of Raspberry PIs. We achieve substantial savings in consumed energy and latency. More precisely, on Raspberry Pi Zero, the propagated signature verification in DNSSEC (vs. traditional DNSSEC validation) reduces latency by a factor of 78x and energy consumption by 47x. For TLS handshake, the advantage to latency and energy by an average of 8x and 8x, respectively (considering the full TLS handshake, which has substantial message delays that are unaffected by CSProp). We also compare CSProp with Elliptic Curve Cryptography (ECC) cipher suite and found that CSProp beats up ECC by 2.7 times. We also study the impact of CSProp on a resource-constrained Arduino based device, where we achieve substantial savings (e.g., 57x and 36x reduction in latency and energy for encryption).

In summary, the paper makes the following contributions:

1. We introduce *Ciphertext and Signature Propagation (CSProp)*, a new cryptographic primitive that allows Public Key Cryptography (PKC) operations at a much lower overhead than traditional implementations.

2. We present a formal definition of the new primitive, as well as an instantiation based on RSA, and prove its security under this construction.

3. We apply CSProp to DNSSEC and TLS and evaluate their performance using experiments on three generations of RaspberryPIs. Our experiments show substantial performance and energy gains from using CSProp (e.g., reducing the latency and energy consumption of DNSSEC by 78x and 47x respectively, and of TLS by 8x for both latency and energy). On a resource-constrained IoT board, the Arduino MKR WIFi 1010, CSProp outperforms RSA public-key operations in latency, power consumption, and memory usage.

We discuss related work in Section 6. We summarize our conclusions and discuss future work in Section 7.

2 Background and Preliminaries

In this section, we present some cryptographic preliminaries to provide the necessary background for describing CSProp. Specifically, we introduce the RSA problem and explain low public exponent RSA. Next, we discuss some special case attacks on RSA with low public exponents. We refer interested readers to [26] for an excellent survey of the subject.

2.1 The RSA Problem

Cryptographic primitives in this work have security based on the RSA problem, which is defined as:

Given integers (N,e) where $N = p \cdot q$ is the product of two secret primes, find d such that $e \cdot d = 1 \pmod{\phi(N)}$, where $\phi(N) = (p-1)(q-1)$ is Euler's totient function.

If $e \cdot d = 1 \pmod{\phi(N)}$ then $x^{e \cdot d} = x \pmod{N}$ and so the modular exponentiation functions $x \mapsto x^e \pmod{N}$ and $x \mapsto x^d \pmod{N}$ are inverses of one another. In cryptography, the stronger assumption is often made that given (N,e) and a random $x \pmod{N}$, it is hard to compute $x^d \pmod{N}$. In cryptographic terminology, this ammounts to saying that $x \mapsto x^e \pmod{N}$ is a *trapdoor permutation*. Moving forward, when we speak about the RSA problem, this is the variant to which we are referring.

2.2 Low Public Exponent RSA

The computation time of RSA encryption and digital signature verification are dominated by the time required to compute the e^{th} power of the message and signature. To reduce computation time, e can be chosen to be a small number. The RSA problem when e is set to a public fixed small value (as opposed to e being chosen randomly in normal RSA) is known as the *low public exponent* variant of RSA[1].

CSProp's use of the small exponent bears some similarities to the use of small public keys in RSA, but with some important differences due to the fact that *CSProp uses a small factor rather than a small full key*. Readers might wonder why not use a full public key that is low (*e.g.,* $e = 3$) for the RSA cryptosystem, which is an idea that has been considered to accelerate RSA operations in the past. Our rationale is two-fold: (1) Security: RSA with low public exponent has been demonstrated to be vulnerable to some types of attacks that could break RSA encryption and verification [27, 36, 54], although they can be prevented by avoiding implementation pitfalls that enable them. In contrast, CSProp is immune against these attacks since we use only a small factor, not the full exponent; and (2) Compatibility: in light of the known attacks on low public exponents, RFC recommendations [38] and vendor practice favor using larger public exponents, which presents a substantial barrier to using low public exponents throughout the system. Despite realization of the potential of using RSA with small public exponents [62] (assuming a secure padding scheme such as RSAES-OAEP [19] and RSASSA-PSS [21] is used), vendors and organizations continue to choose to enforce larger exponents [38]. In contrast, CSProp supports backward compatibility by choosing larger exponents, but requires that *only a factor of the public exponent is low*.

The hardness of the RSA problem and its efficient cousin, RSA with low public exponent, is the subject of an extensive body of work. CSProp is different from traditional low public exponent RSA in that the public exponent consists of two factors in which one of them is small (*i.e.,* not the full public key, which is the exponent e), with important implications that make it *not vulnerable* to some of the attacks on low public exponents. CSProp's resilience to these attacks results is due to: (1) CSProp uses a small public factor, rather than a small public exponent, making its security properties equivalent to RSA before propagation; and (2) Low public exponent security problems arise primarily due to incorrect usage: since the propagation scheme is automated and not typically directly accessible to users, we make sure that it does not have implementation issues. In fact, Section 3 shows that CSProp's security depends on the security of traditional RSA. Next, we review some of published attacks against low public exponent RSA.

Partial Key/Message Exposure Attacks. Coppersmith [36] showed an attack on RSA with low public exponents when the attacker knows two-thirds of the bits of the message. While "message guessing" attacks can easily be avoided if proper padding is used, Boneh, Durfee and Frankel [27] extended Coppersmith's technique to give an attack on RSA with low public exponents when the adversary knows at least a quarter of the bits of the secret key. Other works [18, 50, 87] demonstrate that in some circumstances it is possible to recover bits of the key via side-channel attacks. CSProp is not vulnerable to this attack since the full public exponent is large and only the propagated factor is small (*e.g.,* $e_{\text{low}} = 3$). Thus, the security of CSProp depends on the security of traditional RSA.

Broadcast Attacks. Håstad [54] described a factorization algorithm (thus breaking RSA) if the adversary gets access to 3

[1] Choosing a low private exponent d is insecure and can completely break the cryptosystem [56, 98]

ciphertexts which encrypt the same message under 3 different low public exponent public keys $(N_1,3),(N_2,3),(N_3,3)$. His technique generalizes to larger values of e_{low} and requires roughly e_{low} different encryptions. Other works [11, 37] generalize this method to attack RSA when related (as supposed to the exact same) messages are encrypted multiple times under different low exponent public keys. In our case, since the strength of our propagation procedure (Prop) holds depending on the hardness of the standard RSA ciphertext encryption procedure (Enc), these attacks do not apply to CSProp (see Section §3 for more details).

3 Ciphertext and Signature Propagation

In this section, we formally introduce ciphertext and signature propagation, CSProp. Recall that this primitive provides end-to-end security since it does not need a stateful and trusted proxy. We provide the instantiation of CSProp based on the RSA cryptosystem and present a proof of CSProp's security.

3.1 Definitions

Notation. Throughout this section, we let n denote a security parameter, and let $\mathcal{P}$ and $\mathcal{P}'$ be computational problems representing the cryptographic problems facing the adversary in the original and propagated signature domains respectively.

3.1.1 Signature Propagation

Definition. A *$\mathcal{P}-to-\mathcal{P}'$ signature propagation scheme of rate R* is a set of efficient algorithms:

$$(\mathsf{KeyGen},\mathsf{Sign},\mathsf{Verify},\mathsf{Prop},\mathsf{VerifyProp})$$

satisfying the following syntax, efficiency, correctness, and security requirements.

• **Syntax:**
- KeyGen: this algorithm is used to generate the keys necessary for signature propagation. Its syntax is as follows: $\mathsf{KeyGen}(1^n)$ outputs $(\mathsf{vk},\mathsf{vk}',\mathsf{sk})$.
- Sign and Verify: the syntax for these algorithms is the same as for standard public-key signing and verifying: $\mathsf{Sign}(\mathsf{M},\mathsf{sk},\mathsf{vk})$ outputs σ; and $\mathsf{Verify}(\mathsf{M},\mathsf{vk},\sigma)$ outputs a bit.
- Prop: is used by the stateless and untrusted propagator to generate the propagated signature: $\mathsf{Prop}(\mathsf{M},\mathsf{vk},\mathsf{vk}',\sigma)$ outputs σ'.
- VerifyProp: is used by the client to verify the propagated signature, completing the validation process: $\mathsf{VerifyProp}(\mathsf{M},\mathsf{vk},\mathsf{vk}',\sigma')$ outputs a bit.

• **Efficiency:** We have $R\cdot T' = O(T)$ where T and T' denote the running times of Verify and VerifyProp, respectively.

• **Correctness:** Fix a message M arbitrarily. Consider the random procedure:
1) draw $(\mathsf{vk},\mathsf{vk}',\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^n)$;
2) draw $\sigma \leftarrow \mathsf{Sign}(\mathsf{M},\mathsf{sk},\mathsf{vk})$;
3) draw $\sigma' \leftarrow \mathsf{Prop}(\mathsf{M},\mathsf{vk},\mathsf{vk}',\sigma)$.
Then,

$$\mathsf{Verify}(\mathsf{M},\mathsf{vk},\sigma) = \mathsf{VerifyProp}(\mathsf{M},\mathsf{vk},\mathsf{vk}',\sigma') = 1$$

holds with probability 1.

• **Security:** There are efficient reductions from an adversary who wins the standard existential unforgeability game for $(\mathsf{KeyGen},\mathsf{Sign},\mathsf{Verify})$ (resp. G, below) to an adversary who solves $\mathcal{P}$ (resp. $\mathcal{P}'$). The game G is between a challenger $\mathcal{C}$ and adversary $\mathcal{A}$ and works as follows:

The Signature Propagations Game G:

1. $\mathcal{C}$ draws $(\mathsf{vk},\mathsf{vk}',\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^n)$ and sends $(\mathsf{vk},\mathsf{vk}')$ to $\mathcal{A}$.
2. For $i = 1,\ldots,\mathsf{poly}(n)$: $\mathcal{A}$ sends query messages, M_i, to $\mathcal{C}$; $\mathcal{C}$ computes $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{M}_i,\mathsf{sk},\mathsf{vk})$ and sends σ_i back to $\mathcal{A}$.
3. Finally, $\mathcal{A}$ sends a pair (M^*,σ^*) and wins if:

$$\mathsf{VerifyProp}(\mathsf{M}^*,\mathsf{vk},\mathsf{vk}',\sigma^*) = 1 \text{ and } \mathsf{M}^* \neq \mathsf{M}_i \;\forall\, i.$$

Remark. So in a $\mathcal{P}-\text{to}-\mathcal{P}'$ signature propagation scheme, $(\mathsf{KeyGen},\mathsf{Sign},\mathsf{Verify})$ is a standard signature scheme assuming the hardness of the problem $\mathcal{P}$; and $(\mathsf{KeyGen},\mathsf{Prop}\circ\mathsf{Sign},\mathsf{VerifyProp})$ is a signature scheme assuming hardness of $\mathcal{P}'$; moreover, VerifyProp is $R-$times faster than Verify. Thus, signature propagation gives a way to improve verification efficiency while still maintaining security assuming hardness of $\mathcal{P}'$ (a possibly stronger assumption, which we will demonstrate for RSA).

Propagation for ciphertexts (used for to propagate encryption) is defined similarly.

3.1.2 Ciphertext Propagation

Definition. A *$\mathcal{P}-to-\mathcal{P}'$ ciphertext propagation scheme of rate R* is a set of efficient algorithms:

$$(\mathsf{KeyGen},\mathsf{Enc},\mathsf{Dec},\mathsf{Prop},\mathsf{DecProp})$$

satisfying the following syntax, efficiency, correctness, and security requirements.

• **Syntax:**
- KeyGen: this algorithm is used to generate the keys necessary for ciphertext propagation. Its syntax is as follows: $\mathsf{KeyGen}(1^n)$ outputs $(\mathsf{pk},\mathsf{pk}',\mathsf{sk})$.
- Enc and Dec: the syntax for these algorithms is the same as for standard public-key encryption and decryption: $\mathsf{Enc}(\mathsf{M},\mathsf{pk})$ outputs C; and $\mathsf{Dec}(\mathsf{C},\mathsf{sk})$ outputs a message $\overline{\mathsf{M}}$.
- Prop: is used to generate the propagated ciphertext, completing the encryption: $\mathsf{Prop}(\mathsf{C},\mathsf{pk},\mathsf{pk}')$ outputs C'.
- DecProp: standard public-key decryption is used to decrypt the propagated ciphertext: $\mathsf{DecProp}(\mathsf{C}',\mathsf{sk})$ outputs a message $\overline{\mathsf{M}}$.

• **Efficiency:** We have $R \cdot T = O(T')$ where T and T' denote the running times of Enc and Prop, respectively.

• **Correctness and Security:** $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is a standard encryption scheme assuming the hardness of $\mathcal{P}'$; $(\mathsf{KeyGen}, \mathsf{Prop} \circ \mathsf{Enc}, \mathsf{DecProp})$ is an encryption scheme assuming the hardness of $\mathcal{P}$; correctness and security are inherited.

3.2 Propagating with RSA

In this section, we provide the instantiation of CSProp based on RSA.

3.2.1 Propagating RSA Signatures

We instantiate a $\mathcal{P}-$to$-\mathcal{P}'$ signature propagation scheme where $\mathcal{P}$ is standard RSA and $\mathcal{P}'$ is RSA with low public exponent, specifically with exponent e_{low}. Our construction uses a hash function H, modeled as a random oracle.

- $\mathsf{KeyGen}(1^n)$ generates an RSA modulus $N = p \cdot q$ for secret primes p and q and draws a random e such that $e_{\mathsf{low}} | e$; find d such that $e \cdot d = 1 \pmod{\phi(N)}$. Output:

$$(\mathsf{vk}, \mathsf{vk}', \mathsf{sk}) = \big((N,e),(N,e_{\mathsf{low}}),(N,d)\big).$$

- $\mathsf{Sign}(\mathsf{M}, \mathsf{sk}, \mathsf{vk})$ computes $h = \mathsf{H}(\mathsf{M}, \mathsf{vk})$, and outputs $\sigma = h^d \pmod{N}$.
- $\mathsf{Verify}(\mathsf{M}, \mathsf{vk}, \sigma)$ computes $h = \mathsf{H}(\mathsf{M}, \mathsf{vk})$ and outputs 1 if $\sigma^e = h \pmod{N}$, 0 otherwise.
- $\mathsf{Prop}(\mathsf{M}, \mathsf{vk}, \mathsf{vk}', \sigma)$ outputs $\sigma' = \sigma^{e/e_{\mathsf{low}}} \pmod{N}$.
- $\mathsf{VerifyProp}(\mathsf{M}, \mathsf{vk}, \mathsf{vk}', \sigma')$ computes $h = \mathsf{H}(\mathsf{M}, \mathsf{vk})$ and outputs 1 if $(\sigma')^{e_{\mathsf{low}}} = h \pmod{N}$, 0 otherwise.

Theorem. Let $R = |e|/|e_{\mathsf{low}}|$, where $|e|$ and $|e_{\mathsf{low}}|$ are the bit-lengths of e and e_{low}, respectively. Then,

$$(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Prop}, \mathsf{VerifyProp})$$

is a $\mathcal{P}-$to$-\mathcal{P}'$ signature propagation scheme with rate R, in the random oracle model [20]. The random oracle is a standard strong assumption on perfectly random hash functions supporting the collision resistance property, which we inherit from the use of RSA. Such hash functions require that for every unique input the function generates a unique output chosen with equal probability from the output domain.

3.2.2 Propagating RSA Ciphertexts

We instantiate a $\mathcal{P}'-$to$-\mathcal{P}$ ciphertext propagation scheme, where $\mathcal{P}'$ ($\mathcal{P}$) correspond to the RSA problem with low-public exponent and standard RSA respectively.

- The $\mathsf{KeyGen}(1^n)$ algorithm is the same as for signature propagation. Output: $(\mathsf{pk}, \mathsf{pk}', \mathsf{sk}) = \big((N,e),(N,e_{\mathsf{low}}),(N,d)\big)$.
- The Enc and Dec algorithms are RSA encryption and decryption with low public exponent:
  - $\mathsf{Enc}(\mathsf{M}, \mathsf{pk}')$ outputs $\mathsf{C} = \overline{\mathsf{M}}^{e_{\mathsf{low}}} \pmod{N}$, where $\overline{\mathsf{M}}$ denotes a padded version of M.
  - $\mathsf{Dec}(C, \mathsf{sk})$ computes $\overline{\mathsf{M}} = \mathsf{C}^{de/e_{\mathsf{low}}} \pmod{N}$, and recovers M from $\overline{\mathsf{M}}$.
- $\mathsf{Prop}(\mathsf{C}, \mathsf{pk}, \mathsf{pk}')$ outputs $\mathsf{C}' = \mathsf{C}^{e/e_{\mathsf{low}}} \pmod{N}$.
- $\mathsf{DecProp}(\mathsf{C}', \mathsf{sk})$ computes $\overline{\mathsf{M}} = (\mathsf{C}')^d \pmod{N}$, and recovers M from $\overline{\mathsf{M}}$.

3.2.3 How to choose e_{low}

Choosing the value of e_{low} is an implementation issue that can either be standardized or can be chosen by the origin server. In both cases, the lowest possible exponent recommended is $\mathsf{e} = 3$ [26], but $e = 5$, $e = 17$, and $e = 2^{16} + 1 = 65,537$ are also common. For example, RFC3110 [45] recommends choosing $e = 3$ in order to optimize signature verification in DNSSEC, and Ferguson and Schneier [49] suggest using $e = 3$ for signatures and $e = 5$ for encryption. As discussed in Section 2.2, the security of σ' and C' depends on the RSA assumption with low public exponent which is a widely studied hardness assumption. The consensus in the community is that RSA with low public exponent is a stronger assumption than plain RSA (since any algorithm which breaks RSA would also presumably break RSA with low public exponent). However, RSA with low public exponent is a commonly used assumption. There is no currently known method to break RSA with low public exponent, such a method would be a major breakthrough.

3.3 Security Proof

In this section, we present the security proof of the $\mathcal{P}-$to$-\mathcal{P}'$ signature propagation scheme under the RSA-based instantiation. We omit the security proof for ciphertext propagation since it is analogous. We first prove that the existential unforgeability property of $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ holds, which implies security with respect to signatures. We also discuss the security of the scheme relative to attacks on low public exponents. Finally, using cryptographic game theory, we show that the security of the signature propagation game depends on the security of the standard RSA game only (i.e., CSProp is secure if RSA is secure). Note that the proof is also applied to the instantiation of $\mathcal{P}-$to$-\mathcal{P}'$ ciphertext propagation scheme using RSA.

Proof. Correctness follows from verifying the equation:

$$\big((h^d)^{e/e_{\mathsf{low}}}\big)^{e_{\mathsf{low}}} = (h^{d \cdot (e/e_{\mathsf{low}}) \cdot e_{\mathsf{low}}} = h^{d \cdot e} = h \pmod{N}$$

using $d \cdot e = 1 \pmod{\phi(N)}$. The subroutines Verify and VerifyProp are dominated by computing $e-$th and $e_{\mathsf{low}}-$th powers mod N, which require executing the "square mod N" function $O(|e|)$ and $O(|e_{\mathsf{low}}|)$ times, respectively; thus, the efficiency property holds. Note that $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is the standard RSA signature scheme, except that the exponent e is chosen randomly subject to the condition that $e_{\mathsf{low}} | e$.

This event naturally occurs with probability roughly $1/e_{\mathsf{low}}$ in the plain RSA scheme, and so existential unforgeability of $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ holds since the standard RSA problem is hard [26].

Similarly, attacks on RSA with low public exponent do not apply to our propagation scheme. In particular, because the strength of $\mathsf{VerifyProp}(\mathsf{M}, \mathsf{vk}, \mathsf{vk}', \sigma')$ holds depending on the hardness of the standard RSA signature verification procedure: $\mathsf{Verify}(\mathsf{M}, \mathsf{vk}, \sigma)$ the attacks do not apply to CSProp. In other words, the original signature σ is verified using both exponents e/e_{low} and e_{low}, and so $\mathsf{VerifyProp}(\mathsf{M}, \mathsf{vk}, \mathsf{vk}', \sigma')$ holds *iff* σ' is generated using the propagation procedure: $\mathsf{Prop}(\mathsf{M}, \mathsf{vk}, \mathsf{vk}', \sigma)$. Technically, this means that a malicious proxy that attempts to forge a propagated signature fails by the construction of VerifyProp (Section 3.2) assuming the standard RSA problem to solve subroutine Verify is hard. Note that the proof of security also applies to the RSA ciphertext propagation scheme. Thus, a malicious proxy can cause denial of service but cannot forge a signature on falsified or incorrect data. We elaborate on this case to show how to use an adversary who wins the signature propagation game to solve the RSA problem with public exponent e_{low}, implying the impossibility of this attack provided RSA is secure. The security proof is as follows:

So suppose $\mathcal{A}$ is an efficient adversary who wins the signature propagation game with probability $\varepsilon > 0$. We design another adversary $\mathcal{A}'$ which, given (N, e_{low}) and a random $h^* \pmod N$, outputs $(h^*)^{d_{\mathsf{low}}} \pmod N$ also with probability ε, where d_{low} is such that $d_{\mathsf{low}} \cdot e_{\mathsf{low}} = 1 \pmod{\phi(N)}$. $\mathcal{A}'$ works as follows:

- Upon receiving $(N, e_{\mathsf{low}}, h^*)$, $\mathcal{A}'$ chooses a large random integer e' and sends (N, e, e_{low}) to $\mathcal{A}$ where $e = e_{\mathsf{low}} \cdot e'$.
- Instantiate Q, a set of queries of $\mathcal{A}$ to $Q = \{\ \}$. Each time $\mathcal{A}'$ queries a signature of a message M do:
  - check if M has been asked by $\mathcal{A}$; if so return σ to $\mathcal{A}'$ where (M, σ) is the pair appearing in Q;
  - otherwise, choose a random number $\sigma \pmod N$ and return σ to $\mathcal{A}'$;
  - add (M, σ) to Q;
  - set $h = \sigma^e \pmod N$ and program the input/output pair $\big((\mathsf{M}, N, e), h\big)$ into H, so that if $\mathsf{H}(\mathsf{M}, N, e)$ is computed again at any point in the experiment, h will be returned.
- Finally, when $\mathcal{A}$ is ready to return its forgery of the message M^*, $\mathcal{A}'$ works as follows:
  - if M^* appears as the first coordinate of some pair in Q, $\mathcal{A}'$ aborts giving no output;
  - otherwise, when $\mathcal{A}$ queries H on the input (M^*, N, e), $\mathcal{A}'$ returns h^*;
  - finally $\mathcal{A}$ sends (M^*, σ^*), $\mathcal{A}'$ outputs σ^* and halts.

Notice that $\mathcal{A}'$ answers the queries of $\mathcal{A}$ correctly because $\sigma^e = \mathsf{H}(\mathsf{M}, N, e)$ holds for them all. Furthermore, if h^* is a random number $(\mathrm{mod}\ N)$ then the response to $\mathcal{A}$'s hash query $\mathsf{H}(\mathsf{M}^*, N, e)$ is uniformly distributed. These two observations mean that $\mathcal{A}'$ properly simulates the signature propagation game for $\mathcal{A}$, and so by assumption, $\mathcal{A}$ wins this game with probability ε. Finally, note that whenever $\mathcal{A}$ wins the signature propagation game, $(\sigma^*)^{e_{\mathsf{low}}} = h^* \pmod N$ holds, which implies $\sigma^* = (h^*)^{d_{\mathsf{low}}} \pmod N$, and so $\mathcal{A}'$ breaks low public exponent RSA. ■

4 Applications of CSProp

We illustrate the use and advantages of CSProp on two important Internet protocols: DNSSEC and TLS, which are core protocols with respect to securely connecting end devices to the Internet. Both DNSSEC and TLS are used extensively to provide integrity, confidentiality, and/or authentication for critical data. Such operations are computationally expensive, for instance, Miranda *et al.* [78] analyzes the energy consumption of the Transport Layer Security (TLS) protocol transactions on a mobile device and found that more than 60% of total energy is consumed by TLS overhead. Often real-world configurations force end devices to rely on third parties (*e.g.,* DNS resolver and default gateway) to perform cryptographic functionality such as decryption or verification on their behalf. Although such a setup reduces the requirement on the energy-constrained end devices, it compromises security: if the third party is compromised or spoofed, the end devices are completely compromised. Moreover, the last hop between the third party and the end devices becomes vulnerable to attacks (e.g., a recent client-side attack on DNS bypasses DNSSEC [9]). In this section, we show how we can use CSProp to extend DNSSEC and TLS verification to the end devices, providing security with acceptable overhead.

4.1 CSProp over DNSSEC

The Domain Name System (DNS) is an essential networking protocol. It is responsible for mapping Fully Qualified Domain Names (FQDNs) to their corresponding IP addresses. To defeat certain DNS attacks (*e.g.,* cache poisoning [63] and amplification [1] attacks), DNS SECurity Extension (DNSSEC) [13] is proposed as a form of cryptographic defense to authenticate DNS responses with digital signatures. DNSSEC is standardized by the Internet Engineering Task Force (IETF). Without DNSSEC, DNS becomes vulnerable to different classes of attacks where an attacker attempts to provide false responses to queries [63]. DNSSEC operates by adding cryptographic signatures to existing DNS records to prove that they are legitimate responses from trusted servers. Specifically, these signatures provide DNS clients origin authentication and integrity of data (but not confidentiality). Typically, verification of the signatures is implemented by resolvers, rather than the end devices themselves, to reduce the overhead on the

Table 2: DNSSEC Algorithm Use Statistics

Algorithm		# of DS records Signed	
Code	*Name*	*TLDs*	*Alexa*
3	DSA/SHA1	0	7
5	RSA/SHA-1	163	1305
7	RSASHA1-NSEC3-SHA1	539	5669
8	RSA/SHA-256	2157	10962
10	RSA/SHA-512	37	758
12	ECC-GOST	0	3
13	ECDSAP256SHA256	5	6017
14	ECDSAP384SHA384	0	202

end devices. When an end device performs a DNS query, it sends the query to its resolver. If the data is not present in the resolver's DNS cache, the resolver starts the resolution process by traversing the DNS hierarchy from the root server and down to the corresponding authoritative name server.

Unfortunately, to shield the end devices from these expensive operations, this design leaves opportunities for attackers on the last hop between the resolver and the end device. For example, a resolver that is compromised can arbitrarily falsify information. Moreover, an attacker can spoof the resolver or otherwise inject responses to attack the end devices [9].

Without end-to-end authentication, DNS security cannot be guaranteed. A trivial solution is to ask the end devices to carry out the authentication, but this requires multiple expensive cryptographic operations as discussed in the next section. To secure DNS against attacks [9] we use CSProp to provide low overhead end-to-end DNSSEC validation.

4.1.1 DNSSEC Signing Algorithm

There has been no standardization of a specific zone signing algorithm. The usable algorithms usually appear in DNSKEY, RRSIG, and DS RRsets [14, 58, 85, 97]. In practice, root servers always use **Algorithm 8** (which is RSA/SHA256) [85]. However, to the best of our knowledge, there is no documentation of the algorithm used to sign the zones of TLDs and authoritative name servers. For that, we conducted a measurement study to analyze the DS records of the TLDs by examining the root DNS zone[2]. Similarly, we conducted the measurement study on the top 1 million sites based on Alexa Traffic Rank[3]. As shown in Table 2, we confirm the findings in [85] that **Algorithm 8** is indeed the most widely used algorithm in DNSSEC. Our CSProp protocol supports this algorithm, making it straightforward to deply within the current ecosystem.

4.1.2 Design of DNSSEC with CSProp

DNSSEC using CSProp provides efficient *end-to-end* authentication from the origin server to the end device. CSProp provides signature validation over the entire chain of trust of DNSSEC. The design is illustrated in Figure 2. The components in red represent additions for CSProp. Moreover, in this figure, the end device takes charge of the authentication, whereas in a conventional implementation, the verification traffic is initiated by the resolver. We assume the records are not present at the resolver's cache. We explain the steps in detail as follows:

After the DNS resolution process is completed and before the legitimate response of the requested query (*e.g.,* A record of `www.example.com`) is forwarded to the end device, in step ❶ the DNS resolver receives an RRset of type A along with the corresponding RRSIG record from the authoritative name server (*Auth*). To compute the partial validated signature (RRSIG′) of the above RRsetDNSKEY, the resolver needs the DNSKEY record of $Auth_{\mathsf{ZSK}}$ and sends a query to *Auth* as shown in ❷. In step ❸, *Auth* responds back and sends both $\mathsf{RRsetDNSKEY}_{Auth}$ and the corresponding $\mathsf{RRSIG}_{\mathsf{RRsetA}_{Auth}}$. In step ❹, the resolver computes $(\mathsf{RRSIG}_{\mathsf{RRsetA}_{Auth}})'$ using vk and vk′ of $Auth_{\mathsf{ZSK}}$ and $(\mathsf{RRSIG}_{\mathsf{RRsetDNSKEY}_{Auth}})'$ using vk and vk′ of $Auth_{\mathsf{KSK}}$ and forwards them to the end device along with RRsetA and $\mathsf{RRsetDNSKEY}_{Auth}$. The end device completes the validation process of $(\mathsf{RRSIG}_{\mathsf{RRsetA}_{Auth}})'$ and $(\mathsf{RRSIG}_{\mathsf{RRsetDNSKEY}_{Auth}})'$ using vk′ of $Auth_{\mathsf{ZSK}}$ and $Auth_{\mathsf{KSK}}$, respectively. Then, the end device needs to verify the RRSIG of the DNSKEY record of $Auth_{\mathsf{KSK}}$. As shown in step ❺, it sends a query to the resolver and requests $\mathsf{RRsetDS}_{Auth}$. In step ❻, the resolver forwards the query to the `.com` *TLD* server which responds with $\mathsf{RRsetDS}_{Auth}$ and $\mathsf{RRSIG}_{\mathsf{RRsetDS}_{Auth}}$ as shown in step ❼. To partially verify $\mathsf{RRSIG}_{\mathsf{RRsetDS}_{Auth}}$, the resolver in step ❽ sends a query to the `.com` *TLD* server for the *TLD*'s DNSKEY records. Then, the *TLD* server responds in step ❾ with $\mathsf{RRsetDNSKEY}_{TLD}$ and $\mathsf{RRSIG}_{\mathsf{RRsetDNSKEY}_{TLD}}$.

As in step ❸, the resolver computes in step ❿ $(\mathsf{RRSIG}_{\mathsf{RRsetDS}_{Auth}})'$ using vk and vk′ of TLD_{ZSK} and $(\mathsf{RRSIG}_{\mathsf{RRsetDNSKEY}_{TLD}})'$ using vk and vk′ of TLD_{KSK} and forwards the partial verified RRSIGs to the end device along with $\mathsf{RRsetDS}_{Auth}$ and $\mathsf{RRsetDNSKEY}_{TLD}$. Steps ⓫-⓰ are similar to steps ❺-❿ to verify $\mathsf{RRsetDS}_{TLD}$ and $\mathsf{RRsetDNSKEY}_{Root}$. Finally, the end device compares the $\mathsf{DNSKEY}_{Root_{\mathsf{KSK}}}$ record with the publicly available version, and this completes the DNSSEC validation process.

The ability to establish trust between child and parent zones is an integral part of DNSSEC. We cannot trust any of the DNS records if part of the chain is broken. CSProp over DNSSEC provides complete *end-to-end* protection and secures DNS records from being altered by MitM attackers. Furthermore, the steps of the protocol, and the number of packets exchanged between the parties is the same as in regular DNSSEC with changes isolated to the last hop between the DNS resolver and the end device (in addition to the choice of the public key). These properties make it practical to deploy the design.

[2]The dataset is available online at: https://www.internic.net/domain/root.zone and managed by the Internet Corporation for Assigned Names and Numbers (ICANN)

[3]Alexa Top Sites (ATS) web service: https://aws.amazon.com/alexa-top-sites/

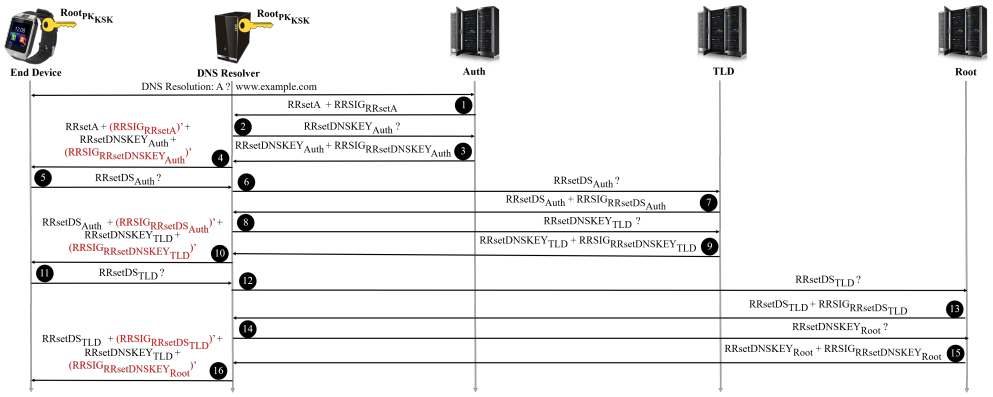


Figure 2: CSProp over DNSSEC — Design

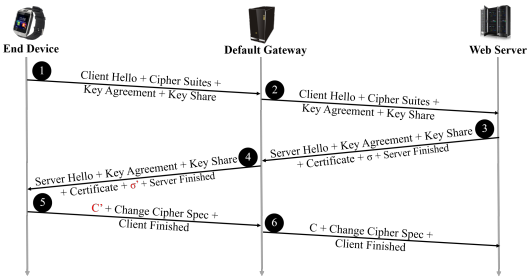


Figure 3: CSProp over TLS — Design

4.2 Optimizing TLS handshakes with CSProp

In the second application, we consider using CSProp to optimize the operation of TLS. The underlying security of TLS protocol relies on the implementation of the cryptographic algorithms during the handshake phase. The cryptographic algorithms provide authentication and integrity between the communicating entities (*i.e.,* in our case, the web server and the end device). To offer these security services, the end device has to handle complex cryptographic operations for validation which are computationally expensive. By using CSProp, we can substantially reduce the computational cost incurred by the handshake phase without compromising security.

TLS is a core security protocol on the Internet and has undergone several revisions over the years to address security and performance flaws specifically in the handshake protocol [83]. We design CSProp to work with TLS 1.3, which is the latest version improving both the performance and security of TLS 1.2. Authenticating the communicating parties to each other is typically done by validating their PKI certificates. The most commonly used certificate is X.509 which is based on the RSA cryptosystem [10]. In common cases, only the web server needs to be authenticated by the client (unless client authentication is required by the server).

CSProp can help reduce the computation cost needed for TLS on the end device by *securely offloading* a considerable part of the encryption and validation processes to the default gateway. Initially, the communication is between the web server and the end device; however, the default gateway is present in typical scenarios of constrained environments (*e.g.,* IoT environment) as shown in Figure 3. The protocol is described in detail as follows:

In step ❶, the end device commences the handshake and sends the "Client Hello" message followed by the cipher suite, key agreement and key share messages to the default gateway in which the latter forwards the messages to the designated web server. In reply, the web server sends in step ❷ the "Server Hello" message comprised of the chosen key agreement, server's X.509 certificate and its associated signature σ, and the server's key share associated with the "Server Finished" message. Then in step ❸, the default gateway forwards all messages received in step 2 to the end device — with one significant change. It substitutes σ with σ' which is the partial verified signature of the server's certificate and σ' is computed using vk and vk'. Now, the end device partially verifies σ' using vk' as shown in step ❹. In step ❺, the end device generates the pre-master secret key K using the web server's key share. K is encrypted using pk' to generate the partial ciphertext C'. The end device sends C', the cipher suite change (if it is applicable) along with the "Client Finished" message to the default gateway. Finally in step ❻, the default gateway completely encrypts K using pk and pk' and forwards messages received in step 5 to the web server along with the full ciphertext C. Upon receiving the messages, the web server using its secret key sk decrypts C to retrieve K, and this concludes the handshake. From here on, all the messages are securely exchanged between the entities.

Similar to CSProp over DNSSEC, CSProp over TLS does not require any additional messages to be exchanged between the three parties that are involved in the handshake phase. This ensures a zero-round trip handshake as in TLS 1.3.

4.3 Propagator Deployment in Practice

CSProp is a general mechanism that can be incorporated within systems with different network and protocol dependent choices for the propagators. For example, it may make sense to have a local uplink router (*e.g.,* a wireless router in a home network or a wireless LAN setting, or a service node on a

Table 3: Experimental Setup and Platforms.

Device	Model	Role	Processor (CPU)	CPU Clock (GHz)	RAM	Cores
Dell	XPS 8700	Origin Server	Intel(R) Core(TM) i7-4790	3.6	16GB	4
Sony VAIO	VPCEA390X	DNS resolver/Default gateway	Intel(R) Core(TM) i5	2.53	8GB	2
Microsoft Surface	Pro 6	End device	Intel(R) Core(TM) i7-8650U	1.9	16GB	4
Raspberry Pi	Zero W	End device (IoT)	ARM11 Broadcom	1	512MB	1
	3 Model B	End device (IoT)	Arm Cortex-A53 (ARMv8)	1.2	1GB	4
	3 Model B+	End device (IoT)	Cortex-A53 (ARMv8)	1.4	1GB	4

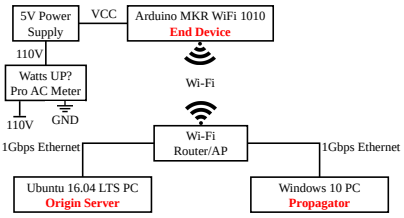


Figure 4: Testbed Architecture Configurations

Radio Access Network in a cellular network setting) serve as a propagator. For more ad hoc network settings, connected nodes or those with larger batteries may serve as propagators, perhaps discoverable using Service Discovery Protocols (e.g., SDP) [71], or reachable using anycast operations [6]. Critically, a man in the middle can only attempt denial of service since any illegal propagation will cause failure of signature verification.

The stateless property of our propagator means that we have the flexibility of changing propagators (*e.g.,* different access points in a wireless LAN as a device moves [61, 65], different road side units [93], or different cluster heads in a sensor network setting [69, 77]). We agree that in a true peer-to-peer setting incentives are a difficult problem to solve especially in the presence of freeloading and Sybil attacks [44].

Key generation distribution for CSProp are similar to their traditional counterparts for PKC ciphers such as RSA. We added an additional field in the exchange packet to include e_{low}, with negligible effect on the packet size.

5 Evaluation

In this section, we experimentally assess the effectiveness of CSProp over DNSSEC and TLS. We compare the protocols under realistic settings and with respect to a number of end devices representative of IoT and embedded devices. We also compare CSProp with ECC cipher suite. We also present a measurement study on a home IoT camera demonstrating the prevelance of operations that use public keys and that can benefit from CSProp.

5.1 Experimental Setup

We evaluate CSProp on four end devices: (1) Microsoft Surface Pro 6; (2) Raspberry Pi Zero W; (3) Raspberry Pi 3 Model B; and (4) Raspberry Pi 3 Model B+ as shown in Table 3. These devices provide a range of embedded/mobile platforms typical of those used in constrained environments [86]. All three Raspberry Pi devices run Raspbian operating system, while Surface Pro 6 runs Windows 10 Home edition operating system. We also evaluate CSProp as a primitive on the Arduino MKR WiFi 1010 [5] as a respresentative of a true constrained IoT device [76]. It uses a 32-bit low power ARM MCU processor (SAMD21) with a clock speed of 48 MHz, 32 KB of SRAM, and 256 KB of flash memory. The hardware acceleration engine for cryptographic algorithms supports the hashing algorithm SHA-256 which we use in our prototype.

We implemented CSProp over DNSSEC and TLS, based on the security library, dnsjava[4], and Bouncy Castle [72], which is a widely used library for cryptography. The dnsjava library is an implementation of DNS in Java and is used by a number of major android applications, such as Netflix, Skype, Samsung Email, and Dailyhunt [42]. For programming the Arduino platform, we use the Arduino-IDE [12] release/v1.8.12, which is the official development framework for Arduino devices. The Arduino prototype is implemented based on the Cryptographic-Protocols-Arduino-and-PC [34] library which has been used by previous work to measure the performance of RSA on IoT devices [66, 90]. We use an Arris router [15] as the gateway communication device. The desktop machines and the Raspberry pis, except Raspberry Pi Zero W, are connected using 1Gbps Ethernet, while the Arduino and the Raspberry Pi Zero W use WiFi.

To measure energy consumption, we use the Watts Up? Pro AC meter [80]. This power meter supports several displays, computer software, and PC interfaces. Its data logger function records all data into non-volatile memory, which we collect to measure the consumed power.

We use a desktop machine running Ubuntu 16.04.6 LTS operating system as an origin server (*i.e.,* DNS servers in DNSSEC, web servers in TLS, and default gateway in IoT environments). For reasons of backward compatibility with middleboxes, we use the recommended key size of 2048-bit and hashing algorithm SHA-256 [68, 83]. We do not consider key and distribution issues which can be difficult at scale. We assume that RSA keys are generated by the origin server and not the end device. We consider the problem of vulnerable keys that are generated by resource-constrained devices [55] to be an orthogonal problem. Our propagator is a desktop machine running Windows 10. Figure 4 shows the main components of the testbed.

We compare CSProp with traditional implementations of DNSSEC validation, TLS handshakes, and RSA public-key operations. We also compare CSProp with current real-world configurations where the public exponent is $2^{16}+1 = 65537$.

5.2 CSProp over DNSSEC

In this section, we show measurement results of CSProp over DNSSEC based on two metrics: (1) Latency; and (2) Energy consumption. We configure a private network (simulating the topology in Figure 1a) to represent the DNS hierarchy. More precisely, we configure the *Root* (.), the *TLD* (`.com`),

[4] Available at: http://www.xbill.org/dnsjava/

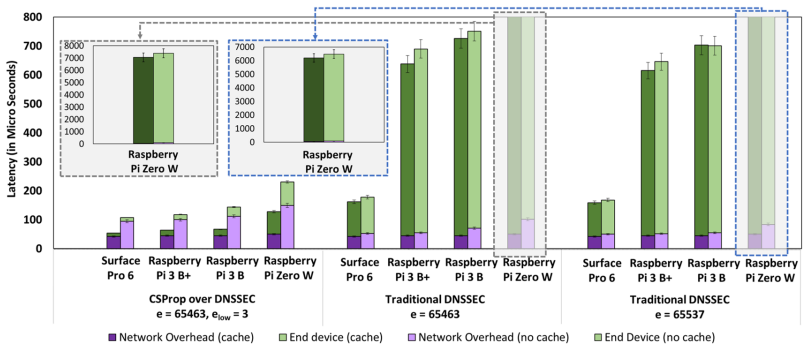


Figure 5: CSProp over DNSSEC — Latency

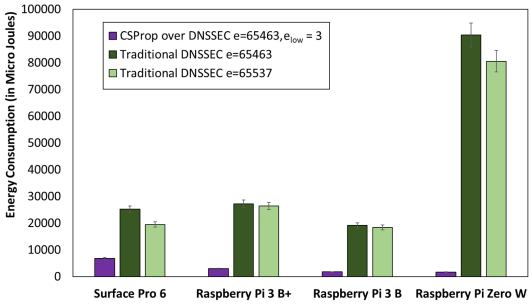


Figure 6: CSProp over DNSSEC — Energy Consumption

and the *Auth* (`example`) name servers internally in the origin server. We use `www.example.com` as the target domain name in our experiments. In addition, the $\text{DNSKEY}_{Root_{\text{KSK}}}$ record (*i.e.,* the trust anchor) is pre-installed at the DNS resolver and all four end devices used in the prototype. To optimize DNSSEC resolution process, the DNS resolver supports the caching property.

We performed the measurements when caching is enabled and disabled at the DNS resolver to get an insight of the impact of caching on the protocol (each experiment is repeated 10 times to bound confidence intervals).

Figure 5 shows a break down of the latency incurred by CSProp over DNSSEC. The latency is broken down into the time consumed by end devices and by the network. The latter time includes: (1) the network overhead caused by sending and receiving packets between the communicated parties; and (2) the time required by the DNS resolver to compute the propagated signature. The results show a significant reduction in latency compared to traditional DNSSEC validation, with a minor impact on latency when the cache is disabled at the DNS resolver. Additionally, we see how device specifications affect performance. For example, in case $e = 65463$ and cache is disabled, we find that CSProp reduces latency by $91x$, $21x$, $35x$, and $10x$ on Raspberry Pi Zero W, Raspberry Pi 3 Model B, Raspberry Pi 3 Model B+, and Surface Pro 6, respectively, compared to traditional DNSSEC validation. Note that the reductions are approximately the same when DNS cache is enabled. We also compared CSProp with current DNSSEC implementations where the used public exponent is 65537. CSProp outperforms this setting, reducing latency by $78x$ on Raspberry Pi Zero compared to conventional DNSSEC when $e = 65537$. The results are marginally better than those when $e = 65463$ in the case of traditional DNSSEC validation, since 65537 is a Fermat number ($2^n + 1$ primes). Fermat numbers are recommended [85] since only the first and last bits of their binary representation are ones (100...001) which minimizes computation cost. Figure 6 shows a significant reduction in energy consumption when CSProp is used; energy is reduced by $53x$, $10x$, $9x$, and $4x$ on Raspberry Pi Zero W, Raspberry Pi 3 Model B, Raspberry Pi 3 Model B+, and Surface Pro 6, respectively.

5.3 CSProp over TLS

Similar to the setup phase with DNSSEC, we configure a private network (with the topology in Figure 1b) where the origin server is a destination web server.We use TLS 1.3 for the handshake phase. The web server's certificate is of type X.509 and is signed by a root CA which its certificate is already pre-installed at the default gateway and end devices. The pre-master secret key (K) is generated using the Advanced Encryption Standard (AES) algorithm as recommended in [83] with 128-bit as the key size.

The latency of the TLS operations are shown in Figure 7. We show the latency incurred by CSProp over TLS but based on the handshake messages: "Client Hello", "Server Hello", and "Client Finished". We use this approach to clearly understand the advantage of our protocol over the existing implementations; specifically when $e = 65537$. CSProp provides $8x$, $4x$, $3x$, and $2x$ reductions in latency (vs. traditional TLS handshake) on Raspberry pi Zero W, Raspberry Pi 3 Model B, Raspberry Pi 3 Model B+, and Surface Pro 6, respectively. We note that these numbers are the full handshake numbers, including the network delays (which are not helped by CSProp).

For energy consumption measurements, we measured the rate at which power is being used at a specific moment in watts (as shown in Figure 8). We found that CSProp, on average, reduces the consumed energy by a factor of $8x$, $3x$, $3x$, and $2x$ on Raspberry Pi Zero W, Raspberry Pi 3 Model B, Raspberry Pi 3 Model B+, and Surface Pro 6, respectively.

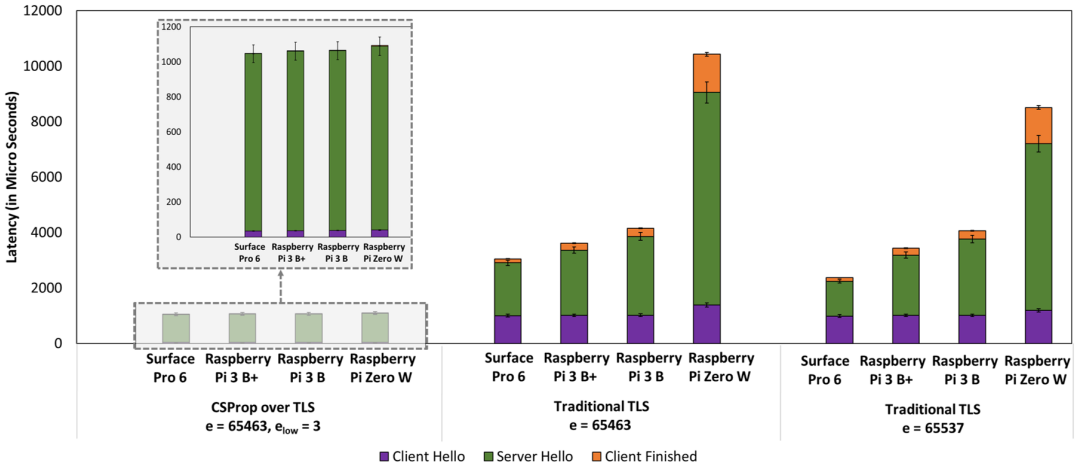


Figure 7: CSProp over TLS — Latency

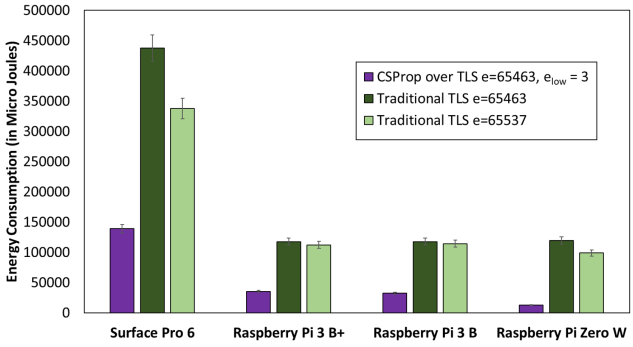


Figure 8: CSProp over TLS — Energy Consumption

Again, these numbers include the energy consumed across the full handshake, with long periods of time taken up for network communication in which the energy consumed is not affected by CSProp.

We note that the less resources the embedded device has, the larger the advantage from CSProp. We conjecture that this occurs since deeply embedded devices are likely not to have energy saving features such as Dynamic Voltage and Frequency Scaling (DVFS) [88], which can help optimize energy efficiency.

5.4 Comparison with Elliptic Curve Cryptography (ECC) Cipher Suites

When power and latency are a consideration, Elliptic Curve Cryptography (ECC) is often considered: it has an approximate equivalent strength to RSA and, in fact, has some advantages relative to using RSA. In particular, key sizes are much shorter: *e.g.,* Elliptic Curve Digital Signature Algorithm (ECDSA) with curve P-256 (which is the standard curve by NIST [7]) has a key size of 256 bits, whereas RSA commonly uses key sizes of 1024 or 2048 bits. Additionally, ECC signatures are much shorter than RSA signatures. However, as mentioned by RFC 6605 [57], even though signing is significantly faster when using ECC than RSA, the opposite is true for signatures validation (RSA is $\simeq 5$ times faster in some implementations). For DNSSEC, this is apparently the most serious challenge when using ECC due to the latency of signature validation. Interestingly, Rijswijk-Deij et al. [96] show that even when using the optimized version of OpenSSL by CloudFlare[5] (in which ECDSA and RSA are sped up by a factor of 8 and 2, respectively), ECDSA is still 6.6 and 3.4 times slower than 1024-bit RSA and 2048-bit RSA, receptively, in terms of signatures validation. More importantly, the actual adoption of ECC by DNSSEC operators is very low [59, 95], raising concerns in regards to backward compatibility if ECC were to be proposed for IoT devices.

Table 4: Comparing CSProp with Elliptic Curve Cryptography (ECC) for TLS handshake latency (*in μ-seconds*)

	Raspberry Pi 3 B+	Raspberry Pi B	Raspberry Pi Zero W
CSProp $e = 65463$, $e_{low} = 3$	1063.24	1066	1093.56
ECDH-ECDSA P-256	2658.18	2984.66	3171.32

For TLS, Gupta et al. [53] conducted a study to analyze the performance of ECC and RSA for SSL (Secure Socket Layer) on resource constrained devices. Their experiments show that TLS handshake using RSA outperform ECC. For completeness, we conducted experimental measurements to compare ECC with CSProp. In our experiments, we used ECDHE-ECDSA (Ephemeral Elliptic Curve Diffie-Hellman key agreement with ECDSA signatures) [79] cipher suite with curve P-256. We run our experiments on three different IoT devices: Raspberry Pi 3 B+, Raspberry Pi B, and Raspberry Pi Zero W (see Table 3 for devices specifications). As shown in Table 4, TLS handshake using CSProp is faster by a factor of $\simeq 2.7x$ than when using ECC. This will impose an additional burden on end devices with the increased CPU load, especially if deployment of ECC-based TLS handshake accelerates. In 2014, Bos et al. [29] surveyed the adoption of ECC and found that only 10% of hosts supported ECC-

[5] https://ripe70.ripe.net/presentations/85-Alg-13-support.pdf

based TLS. On a larger-scale study, the International Computer Science Institute (ICSI) Certificate Notary [60] reported that 11.5% and 2.4% of observed SSL\TLS connections used ECDHE-ECDSA with curves P-256 and P-384, respectively, in June/July 2018. We note also that a variety of attacks on ECC cipher exist [94].

5.5 Performance on Arduino IoT board

Next, we evaluate CSProp on an Arduino MKR WiFi 1010 board, which is a true IoT class system. We were not able to find cryptographic library support to implement the full integration with DNSSEC and TLS. Since we are particularly interested in two RSA public-key operations: verification for signature propagation and encryption for ciphertext propagation, we implement and evaluate the performance for CSProp and traditional RSA public-key operations for these two operations. The measurements reflect the performance of three different RSA key lengths: 512, 1024, and 2048 bits. The code size is $\approx$ 8 KB while the message size (for encryption or verification) is 128 Bytes for all test cases. It is also worth noting that in our implementation we considered basic mathematical operations (*e.g.,* exponentiation and multiplication) without using any optimizations (*e.g.,* montgomery multiplication and optimized squaring as described in [67]). All results are from 50 runs each consisting of 1000 verifications/encryptions in a row to eliminate the code launch/startup effects.

Table 5 summarizes the results of the experiments. For all RSA key sizes, CSProp outperforms the traditional RSA public-key operations in all scenarios. The results show substantial differences in latency, power consumption, and memory footprint; for the same security level, CSProp is clearly a more efficeint alternative for resource-constrained devices. For instance, the execution time for CSProp-encryption and CSProp-verification is 57 and 61 times faster, respectively, compared to traditional RSA encryption for all key sizes. CSProp reduces energy consumption by 36x and 42x for encryption and verification, respectively. Modular exponentiation of CSProp requires little memory (a crucial design decision in designing lightweight cryptosystems) compared to a traditional RSA implementation. More importantly, the results also present interesting findings when different key sizes of the same algorithm are compared. The results show consistent advantage comparing with traditional RSA, making PKI cryptography more practical on resource-constrained environments at all key sizes we considered.

5.6 Importance of Public Key Operations

CSProp can be used to optimize the performance of public key operations. Intuitively, these operations should be common in IoT devices: as a client, rather than a server, it is often verifying signatures of responses from servers. Moreover, as a producer of data, it is often encrypting data that is sent upstream rather than decrypting data. To validate this intuition and study the prevalence of these operations in IoT devices, we analyzed the traffic on an IoT device used in a home environment. Our testbed consists of Wyze Cam V2 (an Amazon choice smart home camera [4]) connected to a wireless home network via an Arris router [15]. The wireless network uses WPA2-AES-128-bit protocol [70] (know as WPA2-Personal) for encryption and a Pre Shared Key (*e.g.,* an 8-character password) for authentication. Our client is a Wyze app downloaded to an iPhone X running iOS 13.3.1. Our results show that the camera uses cryptographic operations continuously. Table 6 shows a trace of collected packets obtained during a live-streaming event for a period of 2-hours, which included more than 60K (*i.e.,* $\approx$ 4.2% of packets exchanges) of RSA public-key operations. Almost exclusively, all operations are public key operations. Although this percentage is small, these operations are substantially more expensive than symmetric key operations and therefore account for a much larger share of the computational power and energy consumed to support. According to benchmarking numbers reported by the eBACS project [2, 3], RSA encryption (a public key operation) requires 2-3 orders of magnitude more time compared to AES encryption; making public key operations cost dominate.

Furthermore, to support end-to-end data protection, we found that transmitted data packets between the camera and the app were frequently encrypted and decrypted using the AES protocol. However, since it is a WPA2-Personal network, this setting secures the network only against outsiders. In particular, this network is vulnerable to Man-in-the-Middle (MitM) attacks if an adversary is an insider who already knows the PSK key. Consequently, she would be able to derive the same secret keys —*i.e.,* Paiwise Transient Key (PTK) and Group Temporal Key (GTK) used to encrypt/decrypt unicast and multicast data packets, respectively, between clients and their associated access point (AP)—that are shared among all users and generated during the 4-Way Handshake protocol [51][6]. We found that $\approx$ 87.5 of data packets are vulnerable to this type of attack. What is worse, in case Domain Name System SECurity Extension (DNSSEC) [13] validation is enabled, more cryptographic operations are required; increasing the computational burden on the IoT camera since chains of DNS RRsets signatures need validation.

6 Related Work

Lightweight cryptography is a term that refers to low overhead cryptographic algorithms designed for energy- or computationally-constrained machines, specially in IoT environments. This is an active area of research in both academia and industry [17, 23, 25, 30, 31, 43, 48, 64, 84, 89].

Symmetric Lightweight Cryptography. With few exceptions, most lightweight cryptography work focuses on symmetric cryptography due to its lower overheads. Lim et al.

[6] Note that using 802.1X [35] for authentication, which is used in WPA2-Enterprise networks, closes this vulnerability. This is because each user is assigned a unique PSK key.

Table 5: Comparison of CSProp VS. traditional RSA public-key operations. Latency is measured in ms, memory footprint in bytes, and energy consumption in mJ. Memory usages for SRAM and ROM are summed for total memory footprint.

	CSProp					
	Encryption			Verification		
Key Size (bits)	Latency (ms)	Memory Footprint (bytes)	EC (mJ)	Latency (ms)	Memory Footprint (bytes)	EC (mJ)
512	11	42	15	15	49	21
1024	29	69	23	35	82	36
2048	61	125	39	71	134	48
	Traditional RSA					
	Encryption			Verification		
Key Size (bits)	Latency (ms)	Memory Footprint (bytes)	EC (mJ)	Latency (ms)	Memory Footprint (bytes)	EC (mJ)
512	634	320	540	915	441	882
1024	1665	552	828	2135	738	1512
2048	3502	1006	1404	4331	1206	2016

Table 6: Profile of the Data Exchanged Between an Iot Device and a Client in a Wireless Home Network

IoT Device	Operation	Total # of Captured Packets	DNS RRsets	RSA					AES 128-bit	
				TLS Handshake	Encryption	Decryption	Signing	Verifying	Encryption	Decryption
WYZE CAM V2 (camera)	Setup	897	68	32	168	0	104	90	232	180
	Pairing	25	2	3	9	0	0	2	5	7
	Live Streaming (2 Hrs)	1460282	108933	29535	30191	0	318	31239	964083	313664

introduce mCrypton [74], following the architecture of Crypton, but reducing key sizes [73]. Similarly, the Scalable Security with Symmetric Keys (S3K) scheme [81], which is a key management architecture, was proposed to provide a scalable energy efficient mechanism to establish trust relationships among entities in IoT environments. These schemes depend on pre-shared keys which might increase the risk to key disclosure and endanger the security services (this is not required in CSProp). Another proposal is Hummingbird [46] which uses a hybrid structure of block and stream ciphers with 16-bit block size and 256-bit key length; an improved version of this work has been developed by Engels et al. [47]. However, Zhang et al. [99] show that the key can be recovered.

Asymmetric Lightweight Cryptography. Relatively fewer efforts target lightweight cryptographic algorithm based on asymmetric ciphers, due to the much higher cost of these operations. Lithe [82] is proposed to provide integration of security between the DLTS protocol at the transport layer and the CoAP protocol at the application layer.The system involves expensive cryptographic processing for both the record and the handshake protocols. In contrast, CSProp requires lightweight cryptographic operations suitable for resource-constrained devices, optimizing TLS handshake latency and energy consumption. Zhang et al. [100] propose a scheme to provide resilience against a large number of sensor node compromises. The scheme incurs lower overhead and higher adaptability than existing techniques that utilize traditional asymmetric algorithms. However, Albrecht et al. [8] show an attack that fundamentally undermines the viability of using perturbation polynomials for designing secure cryptographic schemes. This attack does not apply to our case since the security of CSProp is equivalent to the security of RSA.

Some similar works use cryptographic signature schemes where a powerful server assists in helping a weak client. Bellare and Sandhu [22] consider a group of two-party collaborative RSA signature computation schemes. In contrast, CSProp focuses on efficient signature verification (not generation), which is a public key operation. The proposed protocols follow a similar technique of partitioning the private key into shares. In addition, the security of each protocol in [22] relies on different assumptions on the underlying primitives which make them susceptible to forgery attacks: we showed that CSProp is immune to such attacks in Section 3.3. MacKenzie and Reiter [75] also consider the problem of two-party signature generation. They assume that a user should have and provide a personal password in addition to the split secretkey, making them unsuitable for automated propagation. Damgård and Mikkelsen [40] consider a protocol scenario where four players collaborate to generate a digital signature considering at least one player is malicious. They combine multiple techniques including threshold cryptography signatures where a secret RSA exponent is partitioned between the players. In CSProp, we utilize the RSA public exponent instead of threshold signature. Camenisch *et al.* [33] propose a scheme where a client is authenticated by utilizing a password (similar to [75]), along with a shared secret key split between an end-user and a back-end server. Generating the signing key is initiated at the client side after validating client's password. CSProp neither requires a password nor creates the key at the client side: keys are generated by origin servers. Buldas *et al.* [32] propose a smart-ID scheme where a private exponent is shared between a client and a server for generating signatures. In comparison with CSProp, like the other works discussed here, the scheme is used to optimize private key operations rather than public key operations as with CSProp. We believe our work is the first to enable server aided verification and encryption, which should be useful for weak edge devices in a larger network.

Proxy Assisted Cryptography. Our work bears similarity,

with prior work on proxy-based re-signature schemes, first introduced by Blaze et al. [24] and later revisited by Ateniese and Hohenberge [16]. A significant difference from these works is that CSProp provides security by construction and therefore does not require a trusted proxy to propagate signatures. In contrast, these prior works require a trusted proxy to take a signature as input and generate a new signature as output using the public keys of both parties (the signer and the verifier). We note that this setting is also vulnerable to a known attack on RSA [26, 91, 92].

Joye et al. [62] propose a solution to overcome hardware restrictions enforced by vendors such as Intel Software Guard Extensions (SGX). Although it uses a proxy, the application is different: for CSProp, Patty is helping a weak Alice verify a signature from a more powerful Bob. On the other hand, in this work, Patty is helping a weak Bob sign a message and transmit the signature to a powerful Alice, which is not a common scenario for IoT settings. Critically, Patty needs to know Bob's private key, which is not required in our scheme. For this reason, the warning in [62] that $e'|e$ is problematic does not apply in our setting. This is because e' in [62] is generated using knowledge of Bob's private key, whereas for us e' is computed publicly. Another major difference between this work and ours is the model of security they consider: in addition to the proxy being trusted in this scheme, both public keys must be kept secret, which is incompatible with the requirements of our target applications and the assumptions in our threat model.

7 Concluding Remarks

IoT and embedded devices, in general, are resources-constrained forcing designers to choose either security (e.g., by offloading security to gateway nodes) or performance (performing the expensive cryptographic operations required for end to end security). This paper contributes a new cryptographic primitive, CSProp, that uses a low public exponent to reduce the computational load required by the end devices. We use CSProp to optimize the operation of two core security protocols on the Internet: DNSSEC, and TLS resulting in substantial improvements in latency and energy efficiency. In Section 3.3, we presented the security proof of CSProp using the existential unforgeability property and the cryptographic game theory. One of our future research directions is to implement propagation schemes lattice-based cryptographic signature and encryption. Several recent proposals for lattice-based signature and encryption exist, each one making use of a slightly different hardness assumption and offering slightly different functionality. It would be interesting to try to design propagation schemes to transfer cryptographic content between the schemes so that the schemes with lower overhead/less functionality can be used by the weak computational devices in the network without sacrificing security/functionality for the stronger devices in the network. A limitation of CSProp is that it helps only with operations that use the public key (signature verification, authentication, as well as encryption). For that, other research extensions include integration with support for private key operations to provide a complete solution for PKC in constrained devices; we believe there is substantial need for such support since the vast majority of lightweight cryptography focuses on symmetric ciphers. We also plan to investigate models of propagator deployment and discovery to enable systematic leveraging of CSProp with applications that use PKC.

Acknowledgements

This material is based on work supported by Taibah University (TU) and the Saudi Ministry of Education (MOE). This work is partially supported by the University of California Office of the President UC Lab Fees grant number LFR-18-548554. Any opinions, findings, and conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Dns amplification attacks. *US-CERT*, 2016. Available at https://www.us-cert.gov/ncas/alerts/TA13-088A.

[2] ebacs: Ecrypt benchmarking of cryptographic systems, 2020. Available at http://bench.cr.yp.to/results-encrypt.html.

[3] ebacs: Ecrypt benchmarking of cryptographic systems, 2020. Available at http://bench.cr.yp.to/results-stream.html.

[4] Wyze cam v2 smart home camera, Visited on 2020-03-28. Available at https://www.amazon.com/Wyze-Indoor-Wireless-Detection-Assistant/dp/B076H3SRXG/ref=sr_1_3?dchild=1&keywords=wyze+cam+v2&qid=1585439194&s=electronics&sr=1-3.

[5] Arduino mkr wifi 1010, Visited on 2020-04-01. Available at https://store.arduino.cc/usa/mkr-wifi-1010.

[6] Joe Abley, K Lindqvist, et al. Operation of anycast services. Technical report, BCP 126, RFC 4786, December, 2006.

[7] Mehmet Adalier et al. Efficient and secure elliptic curve cryptography implementation of curve p-256. In *Workshop on Elliptic Curve Cryptography Standards*, volume 66, 2015.

[8] Martin Albrecht, Craig Gentry, Shai Halevi, and Jonathan Katz. Attacking cryptographic schemes based on perturbation polynomials. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 1–10. ACM, 2009.

[9] Fatemah Alharbi, Jie Chang, Yuchen Zhou, Feng Qian, Zhiyun Qian, and Nael Abu-Ghazaleh. Collaborative client-side dns cache poisoning attack. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1153–1161. IEEE, 2019.

[10] Arwa Alrawais, Abdulrahman Alhothaily, Xiuzhen Cheng, Chunqiang Hu, and Jiguo Yu. Secureguard: A certificate validation system in public key infrastructure. *IEEE Transactions on Vehicular Technology*, 67(6):5399–5408, 2018.

[11] P Antonov and V Antonova. Development of the attack against rsa with low public exponent and related messages. In *Proceedings of the 2007 international conference on Computer systems and technologies*, page 50. ACM, 2007.

[12] Arduino. Arduino software, Visited on 2020-04-01. Available at https://www.arduino.cc/en/main/software.

[13] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. Rfc4033:dns security introduction and requirements. Technical report, 2005.

[14] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. Rfc4034:resource records for the dns security extensions. Technical report, 2005.

[15] Arris Router. Watts up pro portable power meter, Visited on 2020-03-28. Available at https://www.amazon.com/NVG468MQ-802-11ac-MoCA%C2%AE2-0-Frontier-Wireless-AC/dp/B073F17BSG/ref=sr_1_1?dchild=1&keywords=Arris+NVG468MQ&qid=1585441528&sr=8-1.

[16] Giuseppe Ateniese and Susan Hohenberger. Proxy re-signatures: new definitions, algorithms, and applications. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 310–319. ACM, 2005.

[17] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A lightweight hash. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–15. Springer, 2010.

[18] Sven Bauer. Attacking exponent blinding in rsa without crt. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 82–88. Springer, 2012.

[19] M Bellare and P Rogaway. Optimal asymmetric encryption padding–how to encrypt with rsa. In *Advances in Cryptology–EUROCRYPT'94*, pages 92–111.

[20] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[21] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 399–416. Springer, 1996.

[22] Mihir Bellare and Ravi S Sandhu. The security of practical two-party rsa signature schemes. *IACR Cryptol. ePrint Arch.*, 2001:60, 2001.

[23] Alex Biryukov and Léo Paul Perrin. State of the art in lightweight symmetric cryptography. 2017.

[24] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 127–144. Springer, 1998.

[25] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. Present: An ultra-lightweight block cipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 450–466. Springer, 2007.

[26] Dan Boneh. Twenty years of attacks on the rsa cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.

[27] Dan Boneh, Glenn Durfee, and Yair Frankel. An attack on RSA given a small fraction of the private key bits. In *Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings*, pages 25–34, 1998.

[28] Dan Boneh and Shay Gueron. Surnaming schemes, fast verification, and applications to sgx technology. In *Cryptographers' Track at the RSA Conference*, pages 149–164. Springer, 2017.

[29] Joppe W Bos, J Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. Elliptic curve cryptography in practice. In *International Conference on Financial Cryptography and Data Security*, pages 157–175. Springer, 2014.

[30] Michael Braun, Erwin Hess, and Bernd Meyer. Using elliptic curves on rfid tags. *International Journal of Computer Science and Network Security*, 2:1–9, 2008.

[31] William J Buchanan, Shancang Li, and Rameez Asif. Lightweight cryptography methods. *Journal of Cyber Security Technology*, 1(3-4):187–201, 2017.

[32] Ahto Buldas, Aivo Kalu, Peeter Laud, and Mart Oruaas. Server-supported rsa signatures for mobile devices. In *European Symposium on Research in Computer Security*, pages 315–333. Springer, 2017.

[33] Jan Camenisch, Anja Lehmann, Gregory Neven, and Kai Samelin. Virtual smart cards: How to sign with a password and a server. In *International Conference on Security and Cryptography for Networks*, pages 353–371. Springer, 2016.

[34] Arpit Chauhan, Inderjit Sidhu, and Archit Pandey. Cryptographic-protocols-arduino-and-pc library, Visited on 2020-04-02. Available at https://github.com/arpitchauhan/cryptographic-protocols-arduino-and-PC.

[35] P Congdon, M Sanchez, and B Aboba. Radius attributes for virtual lan and priority support. *Internet Engineering Task Force, Request for Comment*, 4675:1–13, 2006.

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

[37] Don Coppersmith, Matthew Franklin, Jacques Patarin, and Michael Reiter. Low-exponent rsa with related messages. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–9. Springer, 1996.

[38] Dave Crocker, Tony Hansen, and Murray Kucherawy. Domainkeys identified mail (dkim) signatures. Technical report, RFC 6376, September, 2011.

[39] Tianxiang Dai, Haya Shulman, and Michael Waidner. Dnssec misconfigurations in popular domains. In *International Conference on Cryptology and Network Security*, pages 651–660. Springer, 2016.

[40] Ivan Damgård and Gert Læssøe Mikkelsen. On the theory and practice of personal digital signatures. In *International Workshop on Public Key Cryptography*, pages 277–296. Springer, 2009.

[41] Statista Research Department. Internet of things - number of connected devices worldwide 2015-2025, 2019. Available at https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[42] dnsjava Library Statistics on Andriod. nsjava library statistics on andriod, Visited on 2020-01-09. Available at https://www.appbrain.com/stats/libraries/details/dnsjava/dnsjava.

[43] Qingkuan Dong, Wenxiu Ding, and Lili Wei. Improvement and optimized implementation of cryptogps protocol for low-cost radio-frequency identification authentication. *Security and Communication Networks*, 8(8):1474–1484, 2015.

[44] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[45] D Eastlake 3rd. Rfc3110:rsa/sha-1 sigs and rsa keys in the domain name system (dns). Technical report, 2001.

[46] Daniel Engels, Xinxin Fan, Guang Gong, Honggang Hu, and Eric M Smith. Hummingbird: ultra-lightweight cryptography for resource-constrained devices. In *International Conference on Financial Cryptography and Data Security*, pages 3–18. Springer, 2010.

[47] Daniel Engels, Markku-Juhani O Saarinen, Peter Schweitzer, and Eric M Smith. The hummingbird-2 lightweight authenticated encryption algorithm. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 19–31. Springer, 2011.

[48] Chun-I Fan, Tsung-Pin Chiang, and Ruei-Hau Hsu. Light-weight authentication and key exchange protocols with forward secrecy for digital home. *Journal of Computers*, 18(2):61–74, 2007.

[49] Niels Ferguson and Bruce Schneier. *Practical cryptography*, volume 141. Wiley New York, 2003.

[50] Pierre-Alain Fouque, Sébastien Kunz-Jacques, Gwenaëlle Martinet, Frédéric Muller, and Frédéric Valette. Power attack on small rsa public exponent. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 339–353. Springer, 2006.

[51] IEEE 802.11 Working Group et al. Ieee standard for information technology–telecommunications and information exchange between systems–local and metropolitan area networks–specific requirements–part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 6: Wireless access in vehicular environments. *IEEE Std*, 802(11), 2010.

[52] Shay Gueron. Quick verification of rsa signatures. In *2011 Eighth International Conference on Information Technology: New Generations*, pages 382–386. IEEE, 2011.

[53] Vipul Gupta, Sumit Gupta, Sheueling Chang, and Douglas Stebila. Performance analysis of elliptic curve cryptography for ssl. In *Proceedings of the 1st ACM workshop on Wireless security*, pages 87–94. ACM, 2002.

[54] Johan Hastad. Solving simultaneous modular equations of low degree. *siam Journal on Computing*, 17(2):336–341, 1988.

[55] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 205–220, 2012.

[56] M Jason Hinek. *Cryptanalysis of RSA and its variants*. Chapman and Hall/CRC, 2009.

[57] P Hoffman and W Wijngaards. Rfc 6605: Elliptic curve digital signature algorithm (dsa) for dnssec. internet engineering task force (ietf), 2012.

[58] Paul Hoffman. Rfc6014:cryptographic algorithm identifier allocation for dnssec. Technical report, 2010.

[59] Geoff Huston. Apnic, 2018. Available at https://blog.apnic.net/2018/08/23/measuring-ecdsa-in-dnssec-an-update/.

[60] International Computer Science Institute. The icsi certificate notary, 2018. Available at https://notary.icsi.berkeley.edu/#statistics.

[61] Suman Jana and Sneha K Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. *IEEE transactions on Mobile Computing*, 9(3):449–462, 2009.

[62] Marc Joye and Yan Michalevsky. Rsa signatures under hardware restrictions. In *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security*, pages 51–54. ACM, 2018.

[63] Dan Kaminsky. Black ops 2008: It's the end of the cache as we know it. *Black Hat USA*, 2008.

[64] Masanobu Katagi, Shiho Moriai, et al. Lightweight cryptography for the internet of things. *Sony Corporation*, pages 7–10, 2008.

[65] Vytautas Robertas Kezys. Adaptive beamforming configuration methods and apparatus for wireless access points serving as handoff indication mechanisms in wireless local area networks, March 23 2010. US Patent 7,684,370.

[66] Olha Khomlyak. An investigation of lightweight cryptography and using the key derivation function for a hybrid scheme for security in iot, 2017.

[67] Cetin Kaya Koc. High-speed rsa implementation version 2.0. *RSA Security*, 1994.

[68] Olaf Kolkman, W Mekking, and R Gieben. Rfc6781:dnssec operational practices, version 2. Technical report, 2012.

[69] Dilip Kumar, Trilok C Aseri, and RB2009 Patel. Eehc: Energy efficient heterogeneous clustered scheme for wireless sensor networks. *computer communications*, 32(4):662–667, 2009.

[70] Arash Habibi Lashkari, Mir Mohammad Seyed Danesh, and Behrang Samadi. A survey on wireless security protocols (wep, wpa and wpa2/802.11 i). In *2009 2nd IEEE International Conference on Computer Science and Information Technology*, pages 48–52. IEEE, 2009.

[71] Choonhwa Lee and Sumi Helal. Protocols for service discovery in dynamic and mobile networks. *International Journal of Computer Research*, 11(1):1–12, 2002.

[72] Legion of the Bouncy Castle. Bouncy castle crypto apis, Visited on 2020-01-09. Available at https://www.bouncycastle.org/java.html.

[73] Chae Hoon Lim. Crypton: A new 128-bit block cipher. *NIsT AEs Proposal*, 1998.

[74] Chae Hoon Lim and Tymur Korkishko. mcrypton–a lightweight block cipher for security of low-cost rfid tags and sensors. In *International Workshop on Information Security Applications*, pages 243–258. Springer, 2005.

[75] Philip MacKenzie and Michael K Reiter. Networked cryptographic devices resilient to capture. *International Journal of Information Security*, 2(1):1–20, 2003.

[76] Lukas Malina, Jan Hajny, Radek Fujdiak, and Jiri Hosek. On perspective of security and privacy-preserving solutions in the internet of things. *Computer Networks*, 102, 03 2016.

[77] Vivek Mhatre and Catherine Rosenberg. Homogeneous vs heterogeneous clustered sensor networks: a comparative study. In *2004 IEEE international conference on communications (IEEE Cat. No. 04CH37577)*, volume 6, pages 3646–3651. IEEE, 2004.

[78] Pedro Miranda, Matti Siekkinen, and Heikki Waris. Tls and energy consumption on a mobile device: A measurement study. In *2011 IEEE Symposium on Computers and Communications (ISCC)*, pages 983–989. IEEE, 2011.

[79] Yoav Nir, Simon Josefsson, and Manuel Pegourie-Gonnard. Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls) versions 1.2 and earlier. *Internet Requests for Comments, RFC Editor, RFC 8422*, 2018.

[80] Power Meter Store. Watts up pro portable power meter, Visited on 2020-01-09. Available at https://www.powermeterstore.com/p1206/watts_up_pro.php.

[81] Shahid Raza, Ludwig Seitz, Denis Sitenkov, and Göran Selander. S3k: Scalable security with symmetric keys—dtls key establishment for the internet of things. *IEEE Transactions on Automation Science and Engineering*, 13(3):1270–1280, 2016.

[82] Shahid Raza, Hossein Shafagh, Kasun Hewage, René Hummen, and Thiemo Voigt. Lithe: Lightweight secure coap for the internet of things. *IEEE Sensors Journal*, 13(10):3711–3720, 2013.

[83] Eric Rescorla. Rfc8446:the transport layer security (tls) protocol version 1.3. Technical report, 2018.

[84] Matthew Robshaw. The estream project. In *New Stream Cipher Designs*, pages 1–6. Springer, 2008.

[85] Scott Rose. Rfc6944:applicability statement: Dns security (dnssec) dnskey algorithm implementation status. 2013.

[86] Musa Samaila, Bernardo Sequeiros, AcÃ¡cio Correia, Mario Freire, and Pedro InÃ¡cio. *IoT Hardware Development Platforms: Past, Present, and Future*, pages 107–139. 03 2018.

[87] Werner Schindler and Kouichi Itoh. Exponent blinding does not always lift (partial) spa resistance to higher-level security. In *International Conference on Applied Cryptography and Network Security*, pages 73–90. Springer, 2011.

[88] Greg Semeraro, Grigorios Magklis, Rajeev Balasubramonian, David H Albonesi, Sandhya Dwarkadas, and Michael L Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proceedings Eighth International Symposium on High Performance Computer Architecture*, pages 29–40. IEEE, 2002.

[89] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher clefia. In *International workshop on fast software encryption*, pages 181–195. Springer, 2007.

[90] John C Shovic. Computer security and the iot. In *Raspberry Pi IoT Projects*, pages 213–228. Springer, 2016.

[91] Gustavus J Simmons. A "weak" privacy protocol using the rsa crypto algorithm. *Cryptologia*, 7(2):180–182, 1983.

[92] Gustavus J Simmons. The prisoners' problem and the subliminal channel. In *Advances in Cryptology*, pages 51–67. Springer, 1984.

[93] Sok-Ian Sou and Ozan K Tonguz. Enhancing vanet connectivity through roadside units on highways. *IEEE transactions on vehicular technology*, 60(8):3586–3602, 2011.

[94] Luke Valenta, Nick Sullivan, Antonio Sanso, and Nadia Heninger. In search of curveswap: Measuring elliptic curve implementations in the wild. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 384–398. IEEE, 2018.

[95] Roland van Rijswijk-Deij, Mattijs Jonker, and Anna Sperotto. On the adoption of the elliptic curve digital signature algorithm (ecdsa) in dnssec. In *2016 12th International Conference on Network and Service Management (CNSM)*, pages 258–262. IEEE, 2016.

[96] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. Making the case for elliptic curves in dnssec. *ACM SIGCOMM Computer Communication Review*, 45(5):13–19, 2015.

[97] S Weiler. Rfc3755:legacy resolver compatibility for delegation signer (ds). Technical report, 2004.

[98] Michael J Wiener. Cryptanalysis of short rsa secret exponents. *IEEE Transactions on Information theory*, 36(3):553–558, 1990.

[99] Kai Zhang, Lin Ding, and Jie Guan. Cryptanalysis of hummingbird-2. Technical report, Cryptology ePrint Archive, Report 2012/207, 2012.

[100] Wensheng Zhang, Nalin Subramanian, and Guiling Wang. Lightweight and compromise-resilient message authentication in sensor networks. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 1418–1426. IEEE, 2008.