Securing DNSSEC Keys via Threshold ECDSA from Generic MPC

Anders Dalskov[1], Claudio Orlandi[1], Marcel Keller[2], Kris Shrishak[3](✉), and Haya Shulman[4]

[1] Aarhus University, Aarhus, Denmark
[2] CSIRO's Data61, Sydney, Australia
[3] Technical University Darmstadt, Darmstadt, Germany
kris.shrishak@sit.tu-darmstadt.de
[4] Fraunhofer SIT, Darmstadt, Germany

Abstract. Deployment of DNSSEC, although increasing, still suffers from many practical issues that results in a false sense of security. While many domains outsource zone management, they also have to outsource DNSSEC key management to the DNS operator, making the operator an attractive target for attackers. Moreover, DNSSEC does not provide any sort of protection in the case the operator itself decides to serve false information, for example, if it gets compromised.

In this work, we show how to use techniques from threshold ECDSA: (1) to protect keys such that domains do not reveal their signing keys to a DNS operator, and (2) to protect the operational integrity of DNS operator. As a result of being highly specialized, prior work on threshold ECDSA has focused on a limited set of threat models, and none have so far considered techniques to amortize signature generation. Our work takes a different approach and presents a *generic* technique for obtaining a threshold ECDSA protocol from *any* secure multiparty computation protocol that works over an appropriate finite field. We show how this technique lends itself to very efficient threshold signing protocols by comparing it against state-of-the-art protocols from both academia and industry. For similar threat models, our protocols are as fast as the previous best protocol in terms of signing, and up to an order of magnitude faster for key generation on a fast network. Finally, we show how to integrate our application into a widely used DNS management software and demonstrate through experiments the overhead compared to traditional DNSSECs.

1 Introduction

The Domain Name System (DNS) [RFC1033, RFC1034], one of the core Internet protocols, performs lookup services and provides a platform for an increasing number of systems and applications. DNS was not designed with security in mind and is alarmingly vulnerable to DNS cache poisoning [2,6,9,22,23,36]. DNS Security extensions (DNSSEC) [RFC4033–RFC4035] was standardized to mitigate cache poisoning using cryptographic techniques. At a high level, DNSSEC

L. Chen et al. (Eds.): ESORICS 2020, LNCS 12309, pp. 654–673, 2020.
https://doi.org/10.1007/978-3-030-59013-0_32

enables certification of DNS records in a response such that the machine making the query can verify that it was not tampered with, assuming the DNS operator is trusted. A record is certified using a digital signature scheme, with RSA and ECDSA being the supported algorithms [RFC 5702, RFC 6605]. While RSA is the most commonly used scheme, the need to prevent fragmentation of UDP responses requires the signing keys to be short. ECDSA is the better choice moving forward as it provides the same level of security as RSA but with much smaller signatures. While DNSSEC prevents cache poisoning, this only holds insofar as the operator is trusted. Moreover, DNSSEC additionally requires the operator to manage cryptographic keys. Both these requirements manifest themselves as areas of insecurity in practice.

Centralization of Key Management. DNSSEC burdens the provider with the additional task of generating and managing the keys of their users. Recent work has demonstrated that a large number of domains share the same key [12]. Sharing the same key across multiple domains makes the DNSSEC provider a lucrative target. If the key of one domain is compromised, several *other* domains can be compromised as well. Another study [34] has shown issues with key generation that result in keys with inadequate security.

Centralization of Operation. A second issue that arises from the problem of the DNS operator being in-charge of key management is that the entire operation is centralized. In other words, any guarantee towards integrity of a DNS response to a query is lost if the operator cannot be trusted. This implies that DNSSEC does not prevent attacks from powerful adversaries on the operator, such as nation-state actors. In recent years, several examples of sophisticated attacks on DNS registrars have been observed in Germany [32], Greece [14] and Sweden [30] as part of attacks on DNS infrastructure [37].

1.1 Threshold Signing

Threshold signatures are a natural candidate to solve the issues outlined above. A threshold signature scheme distributes a signing key to n signers such that any subset of at least t signers can sign a message. Since the signing key is distributed among many signers, it will remain private as long as at least t servers remain uncompromised. Moreover, the threshold signing scheme can be made secure against tampering, i.e., a malicious operator cannot compromise the integrity of a response.

While threshold RSA has previously been studied for fault tolerance in DNSSEC, threshold ECDSA has not been used for DNSSEC in spite of an increased interest in threshold ECDSA in recent years [18,19,21,27,28]. All of these recent works motivate the problem of threshold ECDSA in the context of crypto-currencies, a problem that is substantially different from DNSSEC: First, recent work on threshold ECDSA focus on "full threshold", i.e., privacy of the signing key is maintained when up to $t = n - 1$ signers collaborate. Second, the focus has typically been on "malicious" security, i.e., signers are not assumed to behave according to the signing protocol. However, it is possible to design faster protocols by relaxing some of these security guarantees, e.g., by requiring an honest majority, or assuming that signers do not deviate from the signing protocol. The diverse context in which DNS is used can benefit from solutions that are not limited to a specific threat model.

In the real world application of DNSSEC where multiple operators (e.g., 3) serve a domain, the possibility of only one of them being controlled by an adversary is reasonable as operators are often corporations located in different parts of the world and adhere to different local laws. In such a setting, a "full threshold" protocol may not be necessary, and a protocol that assumes an honest-majority (i.e., with 3 servers this implies none of the servers collude) among the operators can be sufficient. Moreover, DNS operators are bound by legal contracts with their customers and they provide service according to this contract. These legal bounds allow us to consider operators that do not act maliciously since that would be a breach of contract. However, in such a case we will still be interested in protecting keys stored at the operator.

1.2 Contributions

A summary of our contributions:

– A generic transformation for secure multiparty computation (MPC) protocols over a field $\mathbb{Z}_p$ to protocols over an elliptic group of order p, such as one used in ECDSA.
– An implementation of this transformation in MP-SPDZ [17] to support threshold signing with ECDSA in many different threat models. We benchmark each instantiation against state-of-the-art protocols for threshold signing and show that they perform comparably.
– A measurement study to understand the extent to which multiple providers for a given domain is used on the Internet.
– A prototype of a full implementation, based on our implementation in MP-SPDZ and Knot, as well as experiments showing that threshold signing incurs only a minimal overhead.

1.3 Outline

Section 2 presents a measurement study we performed. We show that a significant number of domains use multiple operators, which allows them to easily use our solution. Section 3 outlines our system and threat model. Section 4 presents our technical contribution as well as our threshold signing protocol. Section 5 shows how we integrate our signing protocol into a well-known DNSSEC application. Section 6 presents a number of different experiments and comparisons to prior work. Section 7 discusses how our work relates to prior works before the conclusion is presented in Sect. 8.

2 Quantifying Multiple Operators

Since the large DDoS attacks on Dyn [20] and NS1 [5] in 2016, many domains are using more than one operator to increase the redundancy of the zone so that they do not fall victim to another DDoS attack. However, no recent work has measured the number of domains that make use of multiple operators. As we propose to use our multiparty ECDSA protocol for DNSSEC zone signing, we measure the extent to which multiple operators are used on the Internet. We consider a domain to have more than one operator if the DNS name servers of the same domain are hosted by an entirely different DNS operator.

2.1 Data Collection Methodology

If a domain name is configured to be served by three DNS name servers, we check whether it is managed by the same operator. For our purpose, we are interested in nameservers run by different operators and not necessarily name servers placed at different locations. For instance, some domains might use two operators who are geographically located in close proximity to each other; sometimes, even in the same data centre. We are interested in the setting with different operators as they do not trust each other with signing keys and they do not have a business relationship with each other that will allow them to pass on copies of signing keys. Hence, being geographically close does not eliminate the need to run a secure signing protocol. Some domains make use of a single operator which has name servers at different locations. In principle, a multiparty ECDSA protocols can be used in this setting as well because it provides better security than simply storing a copy of the signing key on each name server.

Our measurements were conducted using the Alexa Global Top 1 million list[1] as the dataset. The list was downloaded on 12 July 2019. We ran scans on the same date on all the domains in the dataset and requested its `NS` records. For each `NS` record we also obtain the first associated `A` record. On obtaining the `NS` record, we have the list of authoritative name servers. We compare the sub-domains of *country code TLDs* (ccTLDs), *country code second-level domains* (ccSLDs) and *generic TLDs* (gTLDs). E.g., if the two name servers of a domain are `dns1.p09.nsone.net.` and `ns1.p43.dynect.net.`, then we compare `nsone` and `dynect`. However, we do not only compare the SLD names. For instance, if there is a third name server for the same domain at `pdns6.ultradns.co.uk.`, then we compare `ultradns` with `nsone` and `dynect`.

To measure how many domains use multiple operators, we need to know the owners of the authoritative name servers. Though it is possible to obtain this information from the WHOIS database using the `A` records we collected, the information obtained does not have a consistent schema and is heavily rate limited [29]. Hence, we use the WHOIS database to only check information for Alexa Top-1k; for the rest of Alexa Top 1 million, we take an approach similar to [12] and rely on the `NS` records to indicate the DNS operator. We made manual

[1] https://www.alexa.com/topsites.

checks to make sure that subsidiaries of large corporations are not classified as separate operators. (For instance, Chinese online shopping website `taobao.com` is a subsidiary of the Alibaba group, and we found that one of their name servers is owned by Alibaba and hence, we classified them as the same operator.) Note that large organizations such as Facebook and Google run dedicated networks which provides DNS redundancy. However, as it is run by the same organization, we do not account for them in our list of domains with multiple operators.

2.2 Data Analysis

We classified domains as having a single operator (Only 1), multiple operators (More than 1), no response (NR) and misconfigured (Misconf). An NR classification refers to the case where, during our scans, we did not receive a response with the name server list within a 15-s timeout. Misconf refers to zones which are misconfigured due to mistakes and/or typos. More precisely, we first observed whether we received an `A` record for the `NS` record. If we instead receive an error, we then checked the `NS` record for completeness. If, during this check, we encouter mistakes or typos, the domain is marked as misconfigured. E.g., just `ds0.` was configured as one of the authoritative name servers for the domain `oxfordlearnersdictionaries.com`. See Fig. 1 for the result of classifying the Alexa Global Top 1 million, as well as its subsets, in this manner.

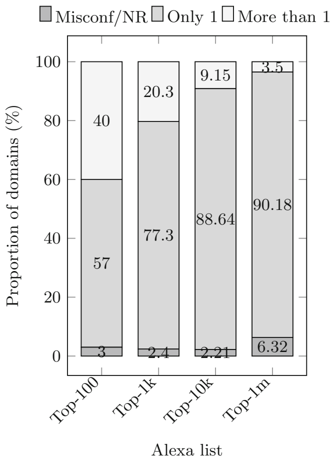

Fig. 1. Domains with multiple operators

We did not receive a response to our queries from 3, 24, 208, 60775 domains in the Top-100, Top-1k, Top-10k and Top-1m respectively. Although we did not find any misconfigured domains in the Top-1k, we found 13 misconfigured domains in the Top-10k and 2483 domains in Top-1m. We observe that 40% of the domains in Alexa Top-100 have more than one operator while the proportion reduces as we move down the Top-1m list. 20.3%, 9.2% and 3.5% of the domains in the Top-1k, Top-10k and Top-1m have more than one operator for their domain. Hence, we conclude from our measurements that there are thousands of domains that use multiple operators and that can easily plug-in our threshold ECDSA protocols.

3 System and Threat Model

The diversity of the DNS ecosystem should be reflected in our system and threat model. For the system model, we assume a small number of operators that serve a single domain. As seen in the previous section, this setting is common in practice, in particular among popular domains. For the threat models, we take the two issues outlined in the introduction as our starting point. Before we continue, we describe the security properties that we address with our threat model:

1. **Key Privacy.** This is our baseline. *Key privacy* states that signing key remains private in the event that a server is compromised. We note that this property is relevant in a number of different contexts. For example, this property states that a signing key isn't exposed to a system administrator, or to anyone who obtains a decommissioned (but improperly cleaned) server.
2. **Operational Integrity.** Besides keeping keys secret, we may also want to uphold the *integrity* of operation. By operational integrity, we mean that only two situations can occur: Either operation proceeds as normal, that is, the right zone is signed, *or* nothing is signed. In other words, at best, a malicious operator can only disrupt operation, i.e., it performs a denial of service attack but it cannot sign zones with bogus information. Notice that key privacy is subsumed in operational integrity. If it is possible to extract the signing key, then no guarantee about the integrity can be made since a single operator can sign any zone it manages at will.

3.1 System and Communication Model

Intuitively, our system model can be viewed as distributing the task of a single operator among multiple operators. To simplify things, we assume that such a system has either $n = 2$ or $n = 3$ operators who maintain a fixed set of domains. These operators can be distributed in a single location, communicating over a LAN, or they can be distributed globally. Finally, we assume that the servers are sufficiently separated, that is, a compromise of one server does not automatically lead to a compromise of another server.

3.2 Threat Model

We consider an adversary that is capable of compromising a single server. Thus, when $n = 2$, the adversary controls half the servers, and when $n = 3$, the adversary controls a minority (since 2 servers remain honest). We distinguish between the two standard adversarial models from the MPC literature. The first type of adversary, called *passive*, is characterized by following the prescribed protocol. The second adversary, called *active*, may behave arbitrarily and not follow the protocol. Notice how these two adversarial types capture our security properties. If we only desire key privacy, then security against a passive adversary suffices. If we want operational integrity as well, then we must also secure ourselves against active adversaries. Indeed, it is exactly against such an adversary that the integrity of operation becomes an issue.

4 Threshold ECDSA

In this section we present a *generic* transformation of any secure computation protocol over a field $\mathbb{Z}_p$ into a protocol for a group of order p. In particular, this technique enables an efficient method to compute threshold ECDSA signatures.

Arithmetic black-box

- A command $([a],[b],[c]) \leftarrow \mathsf{RandMul}()$ that generates appropriate representations of a random tuple of secret shared values $a, b, c \in \mathbb{Z}_p$ with $c = ab$.
- A command $[c] \leftarrow \mathsf{Mul}([a],[b])$ that returns $c = ab$ (This is typically implemented using one invocation of $\mathsf{RandMul}$ and Beaver's re-randomization technique [4]).
- A command $[a] \leftarrow \mathsf{Rand}()$ that generates appropriate representation of a random value $a \in \mathbb{Z}_p$.
- A command $a \leftarrow \mathsf{Open}([a])$ that publicly reconstructs a (or outputs a special symbol $\perp$ denoting abort).
- Linear computation for the $[\cdot]$ representation: given the shares $[a]$, $[b]$ and public scalars $x, y \in \mathbb{Z}_p$, the parties can compute $[c] = x \cdot [a] + y \cdot [b]$ "for free", i.e., the computation does not involve communicating with the other parties.

Fig. 2. The arithmetic black-box functionality.

4.1 ECDSA Signing

ECDSA as standardized in [25] is parametrized by a curve $E(K)$ for a field K. Let $\mathcal{G} \subseteq E(K)$ be an additive subgroup group of order p with generator G, and let $\mathbb{Z}_p$ denote the field of size p. Given a message M, secret key $\mathsf{sk} \in \mathbb{Z}_p$, signing is performed as follows:

1. Sample $k \leftarrow \mathbb{Z}_p$ at random.
2. Compute $(r_x, r_y) = k \cdot G$.
3. Let $s = k^{-1}(H(M) + \mathsf{sk} \cdot r_x)$ where H is a hash function mapping messages into elements of $\mathbb{Z}_p$.
4. Output signature $\sigma = (r_x, s)$.

4.2 Secure Multiparty Computation

We assume a MPC engine supporting the standard commands of the *arithmetic black-box (ABB)* functionality as shown in Fig. 2, where the notation $[a]$ indicates that the value a is "secret-shared", i.e., no party has access to it. The security model of a MPC protocol is parametrized by two variables. First, whether the adversary can control at least half, or less than half the parties. The former is called dishonest majority, while the latter is called honest majority. Observe that an honest majority protocol would correspond a setting with $n = 3$ servers, while a dishonest majority protocol means $n = 2$. The second parameter is the corruption model: The two cases considered here—active and passive—correspond to our description in Sect. 3.2.

4.3 Secure Computation on Groups

We present an extension to the ABB that extends its capabilities to secure computation over an arbitrary Abelian group of order p. In some sense, this shows that the actual representation of the algebraic structure used to perform MPC is irrelevant as long as it is possible to perform linear operations. This generalization of arithmetic MPC has also been described independently by [35], and might have applications in other contexts. In this paper, we use this idea to perform MPC in subgroup $\mathcal{G}$. This extension comes at no extra cost in terms of communication and a small increase in computation complexity (corresponding to standard operations in the subgroup of the curve).

Consider a protocol implementing the ABB in Fig. 2 and assume that the shares $[a]$ are also elements of $\mathbb{Z}_p$. The idea is to let each party map their share of $[a]$ to a curve point of order p by locally computing $A_i = a_i \cdot G$, where a_i is party i's share of a. This mapping, being a homomorphism, preserves linearity and so A_i is a share of $a \cdot G$ with the same properties as the original $\mathbb{Z}_p$ sharing $[a]$. In the following, we write $\langle a \rangle$ to denote a share of $a \cdot G$, and we add the following two commands to the ABB in Fig. 2:

- A command $\langle a \rangle \leftarrow \mathsf{Convert}([a])$ that converts a representation of the shared value a in $\mathbb{Z}_p$ to a representation of the value $a \cdot G$ in the group $\mathcal{G}$.
- A command $a \cdot G \leftarrow \mathsf{Open}(\langle a \rangle)$ that recovers the secret shared point.

These two commands are sufficient to give us a protocol for secure computation over the group $\mathcal{G}$. If we consider the sharing $[a]$ as a vector with elements from $\mathbb{Z}_p$, we get the following useful properties:

- Linearity is preserved, i.e., given the shares $\langle a \rangle$, $\langle b \rangle$ and scalars $x, y \in \mathbb{Z}_p$, we can locally compute $\langle c \rangle = x\langle a \rangle + y\langle b \rangle$.
- If the Open procedure for $[\cdot]$ shares relies only on group operations in $\mathbb{Z}_p$, then we can implement Open for $\langle \cdot \rangle$ shares by using the corresponding group operations of $\mathcal{G}$. This property follows from the fact that $\mathsf{Convert}$ is structure preserving.
- Secret scalar multiplication by public point is possible by noting that $\mathsf{Convert}$ defines an action of $\mathbb{Z}_p$ on $\mathcal{G}$, i.e., $[a] \cdot P$ for a $P \in \mathcal{G}$ is a local operation that results in $\langle a \cdot \log_P(G) \rangle$. Note that opening this share will result in $a \cdot P$.
- Finally, given $[x]$ and $\langle y \rangle$ (and a multiplication tuple $[a], [b], [c]$) it is possible to compute $\langle xy \rangle$ using a slight tweak on Beaver's technique as follows: (1) $e = \mathsf{Open}([a]+[x])$, (2) $D = \mathsf{Open}(\mathsf{Convert}([b])+\langle y \rangle)$, (3) $\langle xy \rangle = \mathsf{Convert}([c]) + e\langle y \rangle + [x]D - eD$. Note that this property is not required for our application but could be of independent interest.

The properties of $\mathsf{Convert}$ and Open, as well as the functionality of the underlying ABB (which provide secure computation over $\mathbb{Z}_p$) is enough to give us a protocol for secure computation over $\mathcal{G}$. This extended ABB (which we will call ABB+) is shown in Fig. 3.

4.4 Active Security Using SPDZ Like MACs

The previous section showed that one can easily extend a protocol of $\mathbb{Z}_p$ with functionality for secure computation over a subgroup of $\mathcal{G} \subseteq E(K)$ of order p.

Extended Arithmetic black-box (ABB+)

- $\mathsf{RandMul}$, $\mathsf{Mul}([\cdot],[\cdot])$, Rand, $\mathsf{Open}([\cdot])$ as described in Figure 2.
- A command $\langle a\rangle \leftarrow \mathsf{Convert}([a])$ that converts a representation of a secret $[a]$ over the field $\mathbb{Z}_p$ into a representation of the secret $\langle a\rangle$ over the group $\mathcal{G}$.
- A command $a \cdot G \leftarrow \mathsf{Open}(\langle a\rangle)$ that reconstructs a curve point $a \cdot G$ from a secret representation $\langle a\rangle$.

Fig. 3. ABB from Fig. 2 extended to support computation over elliptic curves.

A natural question to ask is whether the active security guarantees of the $\mathbb{Z}_p$ protocol extend to the $\mathcal{G}$ protocol. We answer this question in the affirmative by showing that the MAC scheme of SPDZ [16] can be used to provide authentication of shares in $\mathcal{G}$ (i.e., $\langle\cdot\rangle$ shares) as well.

SPDZ Recap. We recall the SPDZ protocol and its security using the description from [15]. In SPDZ a value $a \in \mathbb{Z}_p$ is shared as

$$[a] = ((a_1, \ldots, a_N), (\gamma(a)_i, \ldots, \gamma(a)_N)),$$

where party i holds the pair $(a_i, \gamma(a)_i)$, and where $a = \sum_i a_i$ and $\alpha \cdot a = \gamma(a) = \sum_i \gamma(a)_i$. The value $\alpha \in \mathbb{Z}_p$ is a global MAC key which is secret shared using a different scheme, $[\![\alpha]\!]$. (The details of this are not important for the following discussion; it suffices to say that each party has a share α_i, such that $\sum_i \alpha_i = \alpha$, as well as other information to make this sharing secure) The global MAC key is unknown to all parties and provide a notion of authentication of the shares.

We recap here the opening phase of the SPDZ protocol for a single value, i.e., the part where the parties check if the output was computed correctly[2]:

1. Each P_i has input α_i, their share of the global MAC key, and $\gamma(a)_i$, their share of the MAC on a partially opened value a[3].
2. Each P_i computes $\sigma_i = \gamma_i(a) - \alpha_i a$ and broadcasts a commitment $\mathsf{com}(\sigma_i)$.
3. All parties open $\mathsf{com}(\sigma_i)$, compute $\mathsf{chk} = \sum_i \sigma_i$ and abort if $\mathsf{chk} \neq 0$.

Suppose $a' = a + \epsilon$, i.e., the adversary adds an error $\epsilon \neq 0$ during the partial opening. If, in addition, the adversary lies about its MAC in Step 2 of SPDZ opening phase and let Δ denote this error, then the adversary is successful if $\Delta = \sum_i \sigma_i$. In this case, we have

$$\Delta = \sum_{i=1}^{n} \sigma_i = \sum_{i=1}^{n} \gamma_i(a) - \alpha_i a = \alpha\epsilon.$$

[2] Note that several openings can be batched at the same time, see the original paper [15] for more details.

[3] A partial opening reveals the value but not the MAC.

Since $\epsilon = (a - a') \neq 0$, then $\alpha = \Delta\epsilon^{-1}$ which happens with probability at most $1/p$ due to the random choice of α.

SPDZ-Like Computation Over an Elliptic Curve. In the remainder of this section, we will use the shorthand notation $\mathsf{cv}(a) = \mathsf{Convert}(a)$ interchangeably for convenience. Consider the most natural modification possible to obtain a notion of a SPDZ-sharing $\langle\cdot\rangle$ over $\mathcal{G}$, from a SPDZ-sharing $[\cdot]$ over $\mathbb{Z}_p$, by applying cv to all local shares. We define $\langle a\rangle$ as the vector

$$\langle a\rangle = ((\mathsf{cv}(a_i), \dots, \mathsf{cv}(a_N)), (\mathsf{cv}(\gamma(a)_i), \dots, \mathsf{cv}(\gamma(a)_N))),$$

where P_i holds $(\mathsf{cv}(a_1), \mathsf{cv}(\gamma(a)_i))$. Observe that the linearity of cv implies that $\sum_i(\mathsf{cv}(a_i)) = \mathsf{cv}(\sum_i a_i) = \mathsf{cv}(a)$, which makes the above a valid sharing of $\mathsf{cv}(a)$. In addition, the semantics of the MAC is preserved since

$$\sum_i \mathsf{cv}(\gamma(a)_i) = \mathsf{cv}(\sum_i \gamma(a)_i) = \mathsf{cv}(\alpha \cdot a).$$

Therefore, we can use the same $[\![\alpha]\!]$ to authenticate the $\mathsf{Convert}$ed share as well. More precisely, we consider a modified opening procedure that works as follows:[4]

1. Let α_i be the share of the key held by P_i, and $\varGamma_i = \mathsf{cv}(\gamma(a)_i)$ be the shares of the MAC on $A = \mathsf{cv}(a)$.
2. Each P_i computes $\Sigma_i = \varGamma_i - \alpha_i A$ and broadcasts a commitment $\mathsf{com}(\Sigma_i)$.
3. Open $\mathsf{com}(\Sigma_i)$, compute $\mathsf{chk} = \Sigma_1 + \cdots + \Sigma_N$ and abort if $\mathsf{chk} \neq 0$.

It follows that, due to the linearity of the group operations, if the adversary opens $A' \neq A$, then the check only passes with probability $1/p$. In a nutshell, we are taking a secure linear MAC procedure, and raising all the MACs and values in the exponent. Since the SPDZ MACs are information theoretic secure, the security of the "MAC in the exponent" can be reduced to the security of the regular MAC (as the reduction can run in unbounded time and retrieve the original MAC).

4.5 Multiparty ECDSA Protocol Using the ABB+

We recall the protocol of Gennaro and Goldfeder [21] and show that it can be computed by our extended arithmetic black box functionality. The main issue with computing ECDSA signatures securely is calculating k^{-1} such that it does not reveal information about k. However, the inversion trick by Bar-Ilan and Beaver [3] can be used here: Suppose each party has a share of two random values γ, k, and their product, i.e., $[\gamma]$, $[k]$, $[\delta]$ where $\delta = \gamma \cdot k$. The parties can then open δ and use it locally to compute their share of $[k^{-1}] = \delta^{-1}[\gamma]$. Thus the price to pay for the inversion (which is the most expensive part of every threshold ECDSA protocol) is essentially just generating a random multiplication triple

[4] Once again, the procedure is described for a single value, but it can be extended to support batched opening.

Threshold ECDSA in the ABB+ Hybrid Model

Key Generation. To generate a key for user U_j, either U_j supplies the sharing $[\mathsf{sk}_j]$, or the servers run $[\mathsf{sk}_j] \leftarrow \mathsf{Rand}()$. The public key is computed as $\mathsf{pk}_j = \mathsf{Open}(\mathsf{Convert}([\mathsf{sk}_j]))$.

User independent preprocessing. The goal is to generate a pair $(\langle k\rangle, [k^{-1}])$ for each signature in the following way.

1. The servers run $([a], [b], [c]) \leftarrow \mathsf{RandMul}()$.
2. Run $c \leftarrow \mathsf{Open}([c])$.
3. Let $[k^{-1}] = [a]$.
4. Define $\langle k\rangle \leftarrow \mathsf{Convert}([b]) \cdot c^{-1}$
5. Output $(\langle k\rangle, [k^{-1}])$.

User dependent preprocessing.

1. Take as input $[\mathsf{sk}_j]$ (the sharing of the secret key of user U_j) and $(\langle k\rangle, [k^{-1}])$ (an unused tuple from the previous phase).
2. Compute $[\mathsf{sk}'_j] = [\mathsf{sk}_j/k] \leftarrow \mathsf{Mul}([k^{-1}], [\mathsf{sk}_j])$
3. Output a final tuple $(\langle k\rangle, [k^{-1}], [\mathsf{sk}'_j])$.

Given a message to be signed M and preprocessed tuple $(\langle k\rangle, [k^{-1}], [\mathsf{sk}'_j])$ for U_j.

1. Run $R \leftarrow \mathsf{Open}(\langle k\rangle) = (bc^{-1}) \cdot G = a^{-1} \cdot G = k \cdot G$
2. Let $(r_x, r_y) \leftarrow R$.
3. Compute $[s] = H(M) \cdot [k^{-1}] + r_x \cdot [\mathsf{sk}'_j]$.
4. Open $s \leftarrow \mathsf{Open}([s])$ and output $\sigma = (r_x, s)$.

Fig. 4. Protocol with preprocessing computing threshold ECDSA signatures using our extended ABB.

using $\mathsf{RandMul}$, and using $\mathsf{Convert}$ to compute the value $R = \mathsf{Open}(\mathsf{Convert}([k]))$. The other value we need is a sharing of sk/k. Given $[k^{-1}]$ it is possible to compute $[\mathsf{sk}/k]$ very efficiently by performing a single secure multiplication.

The full protocol using the ABB+ now follows: We consider a setting with a number of servers $\mathcal{S} = \{S_1, \ldots, S_N\}$ and a number of users $\mathcal{U} = \{U_1, \ldots, U_\ell\}$. Our protocol has 4 phases: Key generation in which a random secret key is generated using $[\mathsf{sk}] = \mathsf{Rand}()$, and then converted into the public key by running $\mathsf{pk} = \mathsf{Open}(\mathsf{Convert}([\mathsf{sk}]))$. (Alternatively, users can pick their own keys and input them to the servers in $\mathcal{S}$). Next up are two preprocessing phases: One phase is independent of the users and the messages to be signed, and serves to generate the values $[k^{-1}]$ and $\langle k\rangle$ that are required for generating any signature; the other phase depends on the user and computes $[\mathsf{sk}_j/k]$, where sk_j is the signing key of user U_j. Finally, generating a signature using the output of the preprocessing and the user's signing key is just a matter of performing a linear computation followed by an opening. We show the details of the full protocol in Fig. 4.

Security Analysis. Security of the protocol in Fig. 4 follows directly from the security of the underlying ABB scheme, and from an assumption that ECDSA is a secure signature scheme (this assumption has also been used in [18] and [19]).

5 Multiparty Zone Signing System

In this section, we describe the integration of our threshold ECDSA implementation in a DNS name server before describing the important operations. We implement several variants of our threshold ECDSA protocol on top of MP-SPDZ [17] and have used Crypto++ as the library for computation over elliptic curves. We integrate MP-SPDZ with DNS administrative name servers. For DNS name server software, we use Knot DNS [26] as it has the possibility to perform automated key management and it comes with extensive documentation. For the setting where the registrar is the DNS operator, we propose that registrars interact with other registrars in the zone signing protocol. We describe the multi-operator setting in this section and, where necessary, we note the difference if the operators are also the registrar.

5.1 Setup

In our DNSSEC signing system, each operator serves a name server, runs a threshold ECDSA module and has two key stores: one to store the keys for particular zones and another to store the key material associated with other operators. We consider three name servers operated by independent DNS operators, all of which support ECDSA with SHA256 message digest. We do not change the operation of Knot DNS apart from the parts involved in DNSSEC key generation, key rollover and zone signing. Communication between the name server and the threshold ECDSA module is performed using a message queue.

5.2 Key Generation/Rollover

In the key generation/rollover phase, when new keys need to be generated, each operator generates a signing key sharing $[\mathsf{sk}_j]$ for the zone and runs the key generation as shown in Fig. 4. At the end of this phase, the public key is added to `DNSKEY` record of the zone at all the operators and the signing key share $[\mathsf{sk}_j]$ is stored in the keystore for the zones. In addition, a tag that indicates the DNS operators associated with this signing key share is stored. E.g., Operator A would store a tag $T(B, C)$ along with the key shares associated with Operator B and Operator C. This makes it easy for the threshold ECDSA module to contact the corresponding DNS operators during the signature generation phase. Note that the key generation for ZSK and KSK is the same except that in the case of KSK, the domain owner generates the `DS` record and sends it to the registrar, who then submits it to the registry. When the registrar is one of the DNS operators of the zone, then the registrar can directly submit the `DS` record.

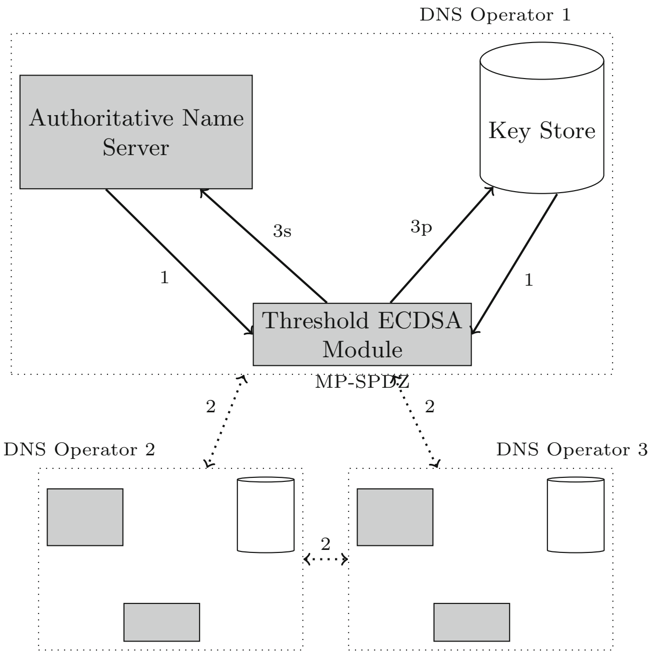

Fig. 5. Zone signing

5.3 Zone Signing

As shown in Fig. 4, our signing protocol has three phases: the first is independent of the zone to be signed, while the second is independent of the RRset, but dependent on the zone to be signed. Each of the three phases involve three steps that are shown in Fig. 5. In Step 1, the threshold ECDSA module receives the input for the phase from the name server and the tag from the key store. In Step 2, the MPC protocol for the phase is run between the threshold ECDSA module of the three operators. In Step 3, the output of the preprocessing phases are sent to the key store (Step 3p) while the output of the signing phase, `RRSIG`, is sent to the name server to store in the zone file (Step 3s). We note that the threshold ECDSA module runs in the background and periodically polls the name server so that it is always available to sign.

Implication for DNS Operators. In our system, the DNS operators do not need to be online any more than they already are in existing systems. DNS operators in existing systems remain online to respond to DNS queries. Many DNS operators sign DNS responses on-the-fly and, hence, they are already equipped with signing systems that are online. In our system they will not only respond to DNS queries, they will also run MPC with other registrar/operators to create `RRSIG`. Our threshold ECDSA protocols have an overhead—both in terms of communication and computation—that depends on the concrete threat and system model. We discuss the overheads as part of our benchmarks in Sect. 6.

It is also worth noting that the operators need not rely on secure hardware to store their user's keys anymore, which may bring down both the cost and complexity for a DNS operator.

Implication for DNS Resolvers. Proper functioning of the DNSSEC ecosystem requires both the signing and the validation to work. Deploying changes at DNS resolvers is extremely hard as numerous resolver software need to be changed. Fortunately, no change is required at the validating resolver to use our solution. Every time the domain is queried at the authoritative name server, the signatures for the zone need to be verified at the resolvers for the chain of trust to be established. Though three operators are involved in the signing process, the signature can be verified with the same `DNSKEY`, irrespective of the operator which initiated the signing process. If the DNS resolver obtains the `DNSKEY` records from Operator A and stores it in the cache, then it will be able to authenticate a response from Operator B for the same domain, as the two operators have the same `DNSKEY` for the zone. The resolver will be able to verify the chain of trust irrespective of the operator responded to the query.

6 Evaluation

In this section we report on several benchmarks of our protocol and compare with prior work of both signature generation and key generation times. We implement six varieties of our protocol (thus supporting different system and threat models) in MP-SDPZ [17]. For $n = 3$ we have *Rep3*, *Shamir* (passive security) and *Mal. Rep3* and *Mal. Shamir* (active security). We remark that only the Shamir protocols support $n > 3$. For $n = 2$, we use *MASCOT* and *MASCOT–* (MASCOT minus) where the latter is a heuristic optimization of the former. Many of these protocols have asymmetric communication patterns and thus we report the maximum execution time, instead of the average. All experiments were run on AWS `c5.2xlarge` instances in three settings: LAN, continental WAN and worldwide WAN. The maximum RTT between any two servers in these settings are 0.08 ms, 17 ms and 240 ms, respectively.

6.1 MASCOT– Optimizations

Our MASCOT– protocol is obtained by making a number of function specific optimizations to MASCOT [24]. Threshold signatures are a special case of MPC where the correctness of the output can trivially be determined by observing the output itself (by verifying the signature). This is a well known trick which has previously been used to optimize many threshold ECDSA protocols in the literature. We can similarly optimize our protocol by using an "optimistic" version of the Open command when running Step 3 of the *Signing* subroutine in Fig. 4.

SPDZ Opening. We save a round of communication during opening as we do not need to check correctness of the MACs. Omitting this attack permits the adversary to make an additive attack, which may result in an invalid signature, but does not leak anything about the secret key.

Beaver Multiplication. Suppose the adversary can perform an additive attack during multiplication. That is, $x + a + \epsilon_1$ and $y + b + \epsilon_2$ for independent ϵ_1 and ϵ_2. A multiplication becomes

$$(x+a+\epsilon_1)\cdot(y+b+\epsilon_2)-(x+a+\epsilon_1)\cdot b-(y+b+\epsilon_2)\cdot a+ab = xy+\epsilon_1 y+\epsilon_2 x+\epsilon_1\epsilon_2.$$

This permits a selective failure attack (e.g., $\epsilon_2 = 0$, $\epsilon_1 \neq 0$ then the multiplication is correct if and only if $y = 0$). However, multiplications are only used on k^{-1} and sk, both of which are of high entropy.

6.2 Comparison with Prior Work

We present a comparison of our protocols with two industry protocols from Unbound [38] and KZen [31], as well as the two-party protocol of Doerner et al. [18] (DKLS) in Table 1. The numbers reported for our protocols correspond to running all three phases in Fig. 4. We see that MASCOT– performs as well as the fastest prior protocol in DKLS, with the same security guarantees, in the LAN setting. However, with more servers, some of our protocols perform better in the LAN setting. In our two WAN settings, DKLS outperforms our protocols, a fact we attribute to the fact that DKLS requires only 2 messages (1 round of communication) whereas our fastest protocol (Rep3) requires 3. Interestingly, the simplicity of our key generation protocol is very apparent, and in all cases (except MASCOT–) key generation is faster than signing.

6.3 Key Generation

We also benchmark the key generation phase as that is typically the more expensive phase in prior works (e.g., [21,28]). With our approach, generating a shared key amounts to running any protocol for generating a secret shared field element $[\mathsf{sk}]$, followed by opening the result of $\mathsf{Convert}[\mathsf{sk}]$. Timings for key generation is shown in Table 2. For our honest majority protocol ($n = 3$) generating a secret key requires only 1 or 2 rounds of communication. MASCOT and MASCOT– is slightly different in that the opening procedure is more costly. Finally, notice that MASCOT– and MASCOT perform the same. Indeed, the heuristics used to obtain MASCOT– cannot be used when generating keys.

6.4 Amortizing Signing

Finally, we analyze the cost of signing when amortization is applied, something that no prior work has considered.[5] Table 3 shows how many signing tuples

[5] Although it might be possible to split some of the protocols in previous work into a preprocessing and signing phase, such a split has not been implemented and, hence, we cannot compare with it.

Table 1. Comparison with prior work. Numbers for our protocols are obtained by taking the mean over the maximum execution time over many runs.

		Colocation		Continent		World	
	n	Sig (ms)	KGen (ms)	Sig (ms)	KGen (ms)	Sig (ms)	KGen (ms)
Rep3	3	2.78	1.45	27.22	29.44	367.87	291.32
Shamir	3	3.02	1.39	78.75	35.52	1140.09	486.82
Mal. Rep3	3	3.45	1.57	82.14	39.97	1128.01	429.47
Mal. Shamir	3	4.43	1.89	174.95	37.35	2340.53	485.11
MASCOT	2	6.56	4.32	196.19	185.71	2688.92	2632.07
MASCOT–	2	3.61	4.41	54.38	181.12	729.08	2654.59
DKLS [18]	2	3.58	43.73	15.33	109.80	234.37	1002.97
Unbound [38]	2	11.33	315.96	31.08	424.02	490.73	1010.98
Kzen [31]	2	310.71	153.87	1282.81	577.67	14441.83	7237.93

Table 2. Breakdown of key generation benchmarks into the time it takes to generate the [sk] sharing, and the time it takes to run Open(Convert([sk])). Times are the maximum time that each step takes.

	Colocation		Continent		World	
	Secret (ms)	Public (ms)	Secret (ms)	Public (ms)	Secret (ms)	Public (ms)
Rep3	0.16	1.27	11.12	18.31	113.86	174.03
Shamir	0.25	1.13	17.17	18.09	243.00	243.82
Mal. Rep3	0.16	1.40	11.00	28.98	115.25	301.66
Mal. Shamir	0.25	1.62	16.90	18.32	241.78	243.18
MASCOT	2.34	1.91	149.26	33.01	2142.31	442.75
MASCOT–	2.40	1.92	145.48	33.21	2132.75	449.43

each protocol can generate per second. The signing times reported in this table correspond to computing a signature when amortization is taken into account. A signing tuple corresponds to the output of the *user dependent preprocessing* phase in Fig. 4. We note that, for almost all protocols, amortized signing corresponds essentially to a single round of communication.

6.5 Overhead for Operators

The storage overhead can be derived from the sizes of a share for a given protocol. For Mal. Rep3, MASCOT and Rep3 each share consists of two $\mathbb{Z}_p$ elements, while for the rest a share is a single element. Thus, for the former three the overhead for storing the signing keys is doubled. A signing tuple consists of two $\mathbb{Z}_p$ shares and one $\mathcal{G}$ share. For example, Rep3 needs to store roughly $2 \cdot 4 \cdot 32$ bytes per signature, assuming a 256-bit prime. Communication per party is between 177 and 354 bytes, depending on the protocol (this number was derived at experimentally).

Table 3. Throughput in signing tuples per second as well as signing time when amortization is taken into account.

	Colocation		Continent		World	
	Tuples per sec.	Sig (ms)	Tuples per sec.	Sig (ms)	Tuples per sec.	Sig (ms)
Rep3	922.27	2.49	898.25	19.91	715.54	247.13
Shamir	1829.69	2.37	1544.31	20.62	402.88	271.80
Mal. Rep3	914.65	2.52	806.13	20.07	309.76	245.14
Mal. Shamir	1792.30	2.91	1154.30	27.03	172.87	416.60
MASCOT	380.19	4.82	233.73	57.02	31.98	756.34
MASCOT–	700.94	2.75	447.85	20.37	68.31	258.85

7 Related Works

DNSSEC Deployment and Measurement. DNSSEC deployment heavily relies on DNS operators and registrars. Prior works have found issues such as reuse of signing keys by DNS operators for multiple domains[6] [12] and sharing of RSA modulus among multiple domains [34]. After the DDoS attacks of 2016, the impact of the attacks and the number of customers of DyN and NS1 that added another operator was measured [1]. However, only the domains that use DyN and NS1 were measured while we measure the use of multiple operators, not restricting our measurements to managed DNS providers.

Privacy in DNS. Though DNSSEC provides data integrity, it does not provide confidentiality. "Range queries" [39] and private information retrieval [40] have been proposed as a solution to hide queries. Recently, the Internet Engineering Task Force (IETF) has considered privacy issues in DNS and DNSSEC [7,8] and proposed DNS-over-TLS [RFC8310] and DNS-over-HTTPS [RFC8484]. While privacy of DNS queries has been considered, we address the issue of privacy of DNSSEC keys.

Threshold ECDSA. Protocols for computing ECDSA signatures in a threshold manner has seen a resurgence lately due to their relevance to crypto-currencies. Doerner et al. have developed threshold ECDSA protocols for both 2-parties [18] and multiple parties [19]. Another recent protocol for dishonest majority is due to Lindell [27]. Even more recently, Castagnos et al. developed a threshold ECDSA protocol from Hash Proof Systems [11].

Threshold Signatures for DNSSEC. Threshold RSA signatures for DNSSEC have been considered in the past. [10] proposed a distributed DNS to avoid single point of failure, which provides fault tolerance and security in the presence of corrupted servers. [13] emulate a HSM at an authoritative name server and they report timings on a LAN which range from tens to hundreds of milliseconds on commodity hardware. Both used RSA threshold signature scheme of [33].

[6] https://www.netnod.se/sites/default/files/2016-12/NETNOD2015_DNS_Martin_Levy_CloudFlare-2.pdf (Slide 28).

8 Conclusion

Deployment of DNSSEC is still an open problem. Current practices force the domain owners to "outsource" management of their DNSSEC keys to the operators, and trust them not to abuse that knowledge. We replace that *trust* with distributed mechanism that generates DNSSEC keys and signatures.

Our mechanism is based on a simple but powerful transformation that can be applied to a large class of protocols for secure computation over $\mathbb{Z}_p$ to obtain protocols for secure computation over an elliptic curve group. We demonstrated the appeal of such a transformation by obtaining several very efficient protocols for threshold ECDSA. Our protocols work in the preprocessing model, which allows us to obtain schemes for computing 100s to 1000s of signatures per second.

Our measurements demonstrate that multi-operator solutions for name servers and for domains are popular in the Internet. Finally, motivated by the aforementioned measurements, we show that our protocols provide an efficient solution to existing issues in DNSSEC. In particular, we demonstrate a system that allows multiple distinct operators to digitally sign zone (as required in DNSSEC) at essentially no cost compared to regular single-operator DNSSEC.

Acknowledgment. This work has been co-funded by: the Concordium Blockhain Research Center, Aarhus University, Denmark; the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC); the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC); the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE; the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation): GRK 2050/251805230 and SFB 1119/236615297.

References

1. Abhishta, A., van Rijswijk-Deij, R., Nieuwenhuis, L.J.M.: Measuring the impact of a successful DDoS attack on the customer behaviour of managed DNS service providers. Comput. Commun. Rev. **48**(5), 70–76 (2018)
2. Atkins, D., Austein, R.: Threat analysis of the domain name system (DNS). RFC **3833**, 1–16 (2004)
3. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, pp. 201–209. ACM (1989)
4. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34
5. Beevers, K.: A note from NS1's CEO: How we responded to last week's major, multi-faceted DDoS attacks, 23 May 2016. https://ns1.com/blog/how-we-responded-to-last-weeks-major-multi-faceted-ddos-attacks
6. Bellovin, S.M.: Using the domain name system for system break-ins. In: USENIX Security Symposium. USENIX Association (1995)

7. Bortzmeyer, S.: DNS privacy considerations. RFC **7626**, 1–17 (2015)
8. Bortzmeyer, S.: DNS query name minimisation to improve privacy. RFC **7816**, 1–11 (2016)
9. Brandt, M., Dai, T., Klein, A., Shulman, H., Waidner, M.: Domain validation++ for MitM-resilient PKI. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 2060–2076. ACM (2018)
10. Cachin, C., Samar, A.: Secure distributed DNS. In: International Conference on Dependable Systems and Networks, 2004, pp. 423–432. IEEE (2004)
11. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Two-party ECDSA from hash proof systems and efficient instantiations. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 191–221. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_7
12. Chung, T., et al.: A longitudinal, end-to-end view of the DNSSEC ecosystem. In: USENIX Security Symposium, pp. 1307–1322. USENIX Association (2017)
13. Cifuentes, F., Hevia, A., Montoto, F., Barros, T., Ramiro, V., Bustos-Jiménez, J.: Poor man's hardware security module (pmHSM): a threshold cryptographic backend for DNSSEC. In: LANC, pp. 59–64. ACM (2016)
14. Cimpanu, C.: Hackers breached Greece's top-level domain registrar, 9 July 2019. https://www.zdnet.com/article/hackers-breached-greeces-top-level-domain-registrar/
15. Damgard, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. Cryptology ePrint Archive, Report 2012/642 (2012). https://eprint.iacr.org/2012/642
16. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_38
17. Data61. MP-SPDZ - versatile framework for multi-party computation. https://github.com/data61/MP-SPDZ
18. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party threshold ECDSA from ECDSA assumptions. In: 2018 IEEE Symposium on Security and Privacy, pp. 980–997. IEEE Computer Society Press, May 2018
19. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Threshold ECDSA from ECDSA assumptions: the multiparty case. In: 2019 IEEE Symposium on Security and Privacy, pp. 1051–1066. IEEE Computer Society Press, May 2019
20. DYN. DYN analysis summary of friday october 21 attack, 26 October 2016. https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/
21. Gennaro, R., Goldfeder, S.: Fast multiparty threshold ECDSA with fast trustless setup. In: ACM Conference on Computer and Communications Security, pp. 1179–1194. ACM (2018)
22. Herzberg, A., Shulman, H.: Socket overloading for fun and cache-poisoning. In: ACSAC, pp. 189–198. ACM (2013)
23. Kaminsky, D.: Black ops 2008: It's the end of the cache as we know it. Black Hat USA (2008)
24. Keller, M., Orsini, E., Scholl, P.: MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 830–842. ACM Press, October 2016
25. Kerry, C.F., Gallagher, P.D.: FIPS pub 186–4 federal information processing standards publication digital signature standard (DSS) (2013)

26. Knot. Knot DNS. https://www.knot-dns.cz/
27. Lindell, Y.: Fast Secure Two-Party ECDSA Signing. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 613–644. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_21
28. Lindell, Y., Nof, A., Ranellucci, S.: Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. Cryptology ePrint Archive, Report 2018/987 (2018). https://eprint.iacr.org/2018/987
29. Liu, S., Foster, I.D., Savage, S., Voelker, G.M., Saul, L.K.: Who is .com?: learning to parse WHOIS records. In: Internet Measurement Conference, pp. 369–380. ACM (2015)
30. Netnod. Statement on man-in-the-middle attack against netnod, 5 February 2019. https://www.netnod.se/news/statement-on-man-in-the-middle-attack-against-netnod
31. Kzen networks. Rust implementation of t, n-threshold ecdsa (elliptic curve digital signature algorithm). https://github.com/KZen-networks/multi-party-ecdsa
32. Krebs on Security. A Deep Dive on the Recent Widespread DNS Hijacking Attacks, 18 February 2019. https://krebsonsecurity.com/2019/02/a-deep-dive-on-the-recent-widespread-dns-hijacking-attacks/
33. Shoup, V.: Practical threshold signatures. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 207–220. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_15
34. Shulman, H., Waidner, M.: One key to sign them all considered vulnerable: evaluation of DNSSEC in the internet. In: NSDI, pp. 131–144. USENIX Association (2017)
35. Smart, N.P., Talibi Alaoui, Y.: Distributing any elliptic curve based protocol. In: Albrecht, M. (ed.) IMACC 2019. LNCS, vol. 11929, pp. 342–366. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35199-1_17
36. Son, S., Shmatikov, V.: The Hitchhiker's guide to DNS cache poisoning. In: Jajodia, S., Zhou, J. (eds.) SecureComm 2010. LNICSSITE, vol. 50, pp. 466–483. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16161-2_27
37. Talos Intelligence. DNS hijacking abuses trust in core internet service, 17 April 2019. https://blog.talosintelligence.com/2019/04/seaturtle.html
38. Unbound Tech. blockchain-crypto-mpc. https://github.com/unbound-tech/blockchain-crypto-mpc
39. Zhao, F., Hori, Y., Sakurai, K.: Analysis of privacy disclosure in DNS query. In: MUE, pp. 952–957. IEEE Computer Society (2007)
40. Zhao, F., Hori, Y., Sakurai, K.: Two-servers PIR based DNS query scheme with privacy-preserving. In: IPC, pp. 299–302. IEEE Computer Society (2007)