Securing DNSSEC Keys via Threshold ECDSA From Generic MPC*

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

[1] {orlandi,anderspkd}@cs.au.dk, Department of Computer Science, DIGIT, Aarhus University
[2] mks.keller@gmail.com, CSIRO's Data61
[3] kris.shrishak@sit.tu-darmstadt.de, TU Darmstadt, Germany
[4] haya.shulman@sit.fraunhofer.de, TU Darmstadt and Fraunhofer SIT, Germany

Abstract. A surge in DNS cache poisoning attacks in the recent years generated an incentive to push the deployment of DNSSEC forward. ICANN accredited registrars are required to support DNSSEC signing for their customers, and the number of signed domains is slowly increasing. Yet with the increase in number of signed domains, the number of vulnerable DNSSEC deployments is also increasing. However, due to lack of support for other, more efficient algorithms, the most popular cryptographic algorithm is RSA. Furthermore, to avoid overhead, the network operators typically use short keys (> 1024 bits) which are no longer considered secure.

In this work, we propose an automated DNSSEC keys generation and zone files signing with threshold ECDSA. We show that a generic transformation suffices to turn essentially any MPC protocol into an equally secure and efficient protocol that computes ECDSA signatures in a threshold setting. The generality of this approach means that DNS operators can pick from a variety of existing efficient MPC solutions which satisfy different security/availability trade-offs. We stress that several of these options were not supported by any previous solution (as a new protocols would have had to be designed for each scenario). We benchmark all the protocols achievable from our transformation. Moreover, as many of the underlying MPC protocols naturally support preprocessing, so does our threshold ECDSA solution (in a way that is independent of both the DNS zone being signed, and the key being used to sign them). We argue that this sort of preprocessing is crucial for pushing deployment of DNSSEC, as it allows DNS operators to sign requests with almost no overhead, compared to the common approach where one operators is completely in charge of their customer's keys.

Depending on the security level and the network configuration, our protocols can preprocess tens, hundreds, or even thousands of signatures per second. Then, the online time for signing essentially matches the RTT for all but the LAN configuration (where signing is still incredibly fast at less than 0.3ms). When comparing with prior work for the same security level, our protocol is never slower and significantly faster in many configurations. For instance, we can generate 4 times as many signatures per second in WAN. Finally, we perform the first study to measure the extent to which multiple DNS operators are used in the Internet and we integrate our novel threshold ECDSA protocols into a DNS application.

1 Introduction

The Domain Name System (DNS) [50,51] is one of the core infrastructures of Internet. DNS resolves human-readable domain names to machine-readable IP addresses over the internet. However, DNS was not designed with security in mind and a number of attacks against DNS, such as DNS cache poisoning and DNS hijacking, have been identified [12,8,42,58,38]. The DNS security extensions (DNSSEC) [3,5,4] are one of the first

* This work was supported by: the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC); the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation program under grant agreement No 803096 (SPEC); the Concordium Blockhain Research Center, Aarhus University, Denmark; the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM); the DFG as part of project D.3 within the RTG 2050 "Privacy and Trust for Mobile Users" and project S3 within the CRC 1119 CROSSING, the German Federal Ministry of Education and Research (BMBF), by the Hessian Ministry of Science and the Arts within CRISP.

attempts at securing DNS against such threats using cryptographic techniques. DNSSEC aims at providing *origin authentication* and *integrity* of DNS data. That is, users can verify whether the response to their DNS queries have originated from the intended DNS server, and that it remained unaltered during transit.

In practice, very few domain owners run their own authoritative name servers and manage their zones, and this role is usually outsourced to the DNS operators. Outsourcing management of zones comes with several benefits, such as increased availability and a lesser chance of misconfiguration (presumably, because the operators have specialists at hand that administer their name servers). However, when DNSSEC is used, this outsourcing also introduces several issues related to key management. In order to use DNSSEC, the operator would need to handle domain owners' private signing keys. This not only means that each domain owner needs to relinquish control of their signing keys, but it also makes the operator a very lucrative target for attackers. Access to the signing keys gives the operator full access to the zone, that is, the operator can change the zone file at will and include malicious information. Assuming that the domain owner trusts the operator, there are two situations that need to be considered. (1) For operational reasons, the operator may reuse keys for multiple domains. In fact, [19] find that 132, 000 domains in their measurements share the same keys. If a single key is compromised, all the domains that share this key will be affected. (2) Consider a situation where a powerful adversary forces the operator to disclose the signing keys of the domains it manages. This is not a hypothetical situation. Depending on the country of operation, governments can legally compel the registrar to behave according their orders and take down domains. It is possible that compromised keys can result in a complete takeover, for instance, of a top-level domain (TLD) [53]. E.g., access to the private part of the ZSK allows an adversary to change the `DS` record of a child zone and validly sign them. Such an access gives the adversary the capability to make changes that will result in a failed verification of the chain of trust [6].

Furthermore, recent large distributed denial of service (DDoS) attacks such as the ones aimed at Dyn [30] or NS1 [11] have illustrated that there is a real need to distribute management of zones to multiple operators. However, the situation with respect to using DNSSEC in a multi-operator setting is currently complex. It is challenging to deploy DNSSEC in an environment where domain owners use multiple DNS operators to distribute their authoritative DNS service. Despite these challenges, recent efforts have been made to address using DNSSEC in a setting with multiple operators [41]. The main issue identified by this RFC is key management, that is, who should be in charge of the signing keys.

1.1 Threshold Signatures

Threshold signatures is a technique wherein a set of n participants jointly compute a signature on a message in such a way that at least $t + 1$, with $t < n$, are required to participate. Threshold signatures are, in a nutshell, a specific kind of secure multiparty computation (MPC) with applicability to many practical problems—in particular DNSSEC.

It is easy to see why threshold signatures would be beneficial for DNSSEC: using such a protocol, no single operator needs to be responsible for the entire signing key. In fact, the domain owner can keep the key themselves and simply provide shares of the key to the operators, where the shares themselves provide no information about the signing key (unless more than t operators band together). Using only these shares, the operators can then digitally sign the zone files, and if the domain owner trusts $t + 1$ of the operators, it is clear that only valid zone files are ever signed.

Therefore, using threshold signatures as a way to distribute the trust in a multi-provider DNSSEC setting seems like a natural approach. This has previously been studied with threshold RSA as the signing scheme [16], however, not with threshold ECDSA, which is the topic of this work.

1.2 Contributions

While schemes for threshold signatures were introduced at least a couple of decades ago, it is only recently that practical schemes for ECDSA have been developed. The renewed interest in threshold signatures, and threshold ECDSA in particular, is mostly due to its use in cryptocurrencies. However, using threshold

signatures in the context of cryptocurrencies is substantially different from their deployment in the context of DNSSEC: First, there is only a handful of DNS operators that are typically used per zone in the world of DNS, while the identity and number of parties involved in cryptocurrencies can vary to a much higher degree. Second, when DNS operators use online signing, they need to generate signatures on-the-fly and hence, many signatures need to be generated per second, especially at large DNS providers. This setting differs from cryptocurrency transactions where very few signatures need to be generated over time (and therefore amortization has only been considered to a lesser degree).

It is thus natural to ask whether it is possible to depart from the design of existing threshold ECDSA protocol and construct protocols that better suits the requirements of the application that we have in mind.

In this work, we answer this question in the positive by describing a very generic and powerful transformation that can turn essentially any MPC protocols for arithmetic circuits over a field $\mathbb{Z}_p$ to a protocol over any other group $\mathcal{G}$ of the same order. Naturally, this allows us to perform MPC in the subgroup $\mathcal{G} \subseteq E(K)$ of order p of the curve $E(K)$ used in ECDSA; in particular, this transformation allows us to transform an *efficient* protocol over $\mathbb{Z}_p$ into an essentially *equally efficient* protocol for curve arithmetic.

Due to the generality of our approach, we are able to obtain several different instantiations of threshold ECDSA for different corruption models (honest or dishonest majority and passive or active adversaries) that can be used to compute a large amount of signatures at a very low cost. Another consequence of the generality of our approach is that key generation becomes very efficient. This too is beneficial in our setting, as large operators may need to generate many keys as they manage many zones. Moreover, it also makes our approach very attractive for other applications, such as cryptocurrencies, as key generation turns out to be a very expensive step in previous protocols.[5]

Finally, we describe an application based on Knot [44] that employs our novel threshold ECDSA protocol for the purpose of zone file signing.

Summary of our contributions:

- We present a generic transformation for MPC protocols over $\mathbb{Z}_p$ to protocols for performing MPC over elliptic curves. This transformation only introduces a small overhead in terms of efficiency and preserves the security properties of the original MPC protocol. We believe that this transformation might have other useful applications beyond ECDSA signing.
- We extend MP-SPDZ [25], one of the main frameworks for MPC, with our transformation. This gives rise to threshold ECDSA protocols for a variety of configurations, many which had never been implemented before in the literature. This illustrates the generality and ease of use of our transformation.
- We benchmark all our instantiations and compare with the state of the art for threshold ECDSA. For the setting of dishonest majority, active security, the signing phase is 16 times as fast in the LAN setting (while in WAN both our protocol and previous work essentially match the RTT between the different servers). The preprocessing phase of our protocol allows us to precompute 4 times as many signatures in the WAN setting (while in the LAN setting the number of preprocessed signatures is essentially the same as prior work).
- For three parties with an honest majority (for which no prior solutions are available), our approach preprocesses more than 600 signatures per second, and signing a single message takes around 240ms when servers are placed on three different continents (North America, Europe and East Asia, respectively). In a colocated setting, we can process up to 1400 signatures per second, with a single signature taking around 0.2ms to create.
- We perform the first measurement study to understand the extent to which multiple operators are used in the Internet. We use Alexa Top-1m dataset to perform our measurements. We find that 40% of the domains in the Alexa Top-100 use multiple operators while the proportion of domains in the Alexa-1m is 3.5%. Our measurements also show that there are thousands of domains that use multiple operators where our threshold ECDSA protocols can be easily used.

[5] Remember that, in order to enhance anonymity, users in classic cryptocurrencies such as Bitcoin or Ethereum are encouraged not to reuse addresses, and therefore, key generation is a much more frequent operation than one might expect.

– We describe a full system that can be used by multiple DNS operators for the purpose of implementing the DNSSEC standard, such that no operator has access to the signing key, and where the overhead is minimal; essentially, the same as sending a single message between all operators.

Concurrent Work. The core technical idea behind this work has been independently developed by Smart and Talibi Alaoui [57]. Here is a detailed comparison of the two works: both works observe that generic arithmetic MPC tools can be used to perform efficient MPC over elliptic curves. Both works include a description of how SPDZ-style MACs can be adapted to elliptic curve computation, and notice that the same idea can be used for other arithmetic MPC protocols (e.g., Shamir- and replicated- secret-sharing). Both works mention threshold ECDSA as a natural application, but we in addition split the protocol in user-independent preprocessing and user/message-dependent computation, as we imagine many users outsourcing their signing capabilities to the same subset of semi-trusted servers. The main differences are: we have implemented and benchmarked the resulting protocols against state of the art threshold ECDSA protocols, and we demonstrate how this can be effectively integrated into existing DNS infrastructure. They do not report on any implementation, but instead describe how the same technique can also be used to perform MPC of a shuffle network.

2 Background

The following section provides an overview of DNSSEC and the current state-of-the-art for threshold ECDSA protocols.

2.1 DNSSEC

DNSSEC [3,5,4] is designed to overcome some of the security limitations of DNS. It provides *origin authentication* and *integrity* of DNS resource record sets (RRsets) by digitally signing them. For this purpose, it introduces three new resource records—`DNSKEY`, `RRSIG` and `DS`. `DNSKEY` record holds the public keys whose corresponding private key is used to sign RRsets. `DNSKEY` is used to verify these signatures. Each zone usually creates two DNSKEY records—one for a Key Signing Key (KSK) and another for a Zone Signing Key (ZSK). The private key of the KSK is used to sign `DNSKEY` records, and the private key of the ZSK is used to sign all other records. The use of two key pairs facilitates frequent change of ZSKs. The process of changing the public/private key pairs in DNSSEC is called *key rollover*. `RRSIG` or Resource Record Signature hold the signature of RRsets. `DS` or Delegation Signer contains the hash of the public key (`DNSKEY`) of the child zone. It is uploaded to the parent zone by the registrar. The `DS` records are signed by the parent zone. It establishes a chain of trust between the parent and the child zone.

DNSSEC ecosystem. DNSSEC ecosystem has two parts: The signing part and the validation part. Both parts are necessary for DNSSEC to function. On the signing part, there are three main organizations: Registries, registrars and DNS operators. *Registries* are organizations that manage top-level domains (TLDs) and they are responsible for maintaining TLD zone file. A domain owner owns a domain name and registers the domain with a *registrar*. The domain owner is a customer of the registrar. *DNS operators* are organizations that run authoritative DNS servers. Each domain name has one or more operators while each operator can serve many domains. DNS operators decide the life cycle of the keys as well as signatures. this is important as DNSSEC does not have the possibility to revoke keys and instead relies on generating new keys and signatures periodically. On the validation part, *DNS resolver* queries DNS authoritative name servers to map domain names to IP addresses. DNS resolver also verifies the signatures (`RRSIG`) for the corresponding RRset. Furthermore, DNS resolvers check the global chain of trust from the root of DNS to the queried domain. The domain is resolved when the signature verification and the global chain of trust is checked.

Role of Registrars. Some of the registrars offer their customers the option where (1) the domain owner can operate the domain, or (2) the registrar becomes the DNS operator and serves as the authoritative name server for the domain being purchased, or (3) a third-party operates becomes the DNS operator. With regards to DNSSEC, the registrar plays a critical role irrespective of the option chosen by the domain owner. When the registrar registers its customer's domain name with the registry, it adds the `NS` record as well

as DS record to the TLD zone file. Typically, only the registrar is allowed to modify information about the domain names it has registered at the registry. Note that in the case of a third-party as a DNS operator, the third-party cannot ask the registrar to upload the DS record. Instead, it has to ask the domain owner to relay the DS record to the registrar. As only the registrar can upload a DS record for the domain to the registry and a missing DS breaks the chain of trust, the registrar plays a critical role in the DNSSEC ecosystem.

Multi-operator setting. The evolution of DNS over the past two decades has seen the move from the recommendation to swap secondary zones with other organizations [31] to the reliance on DNS operators with global points of presence. Nevertheless, many domains were not accessible in the face of large-scaled distributed denial of service attacks, e.g., on NS1 [11] and Dyn [30] in 2016, because of their reliance on a single DNS operator. One solution to safeguard against such attacks on the infrastructure of a single operator is to use multiple DNS operators to host zone files. If the zone is signed by one signing server and only served by the other operators as a secondary authoritative name server, then standard zone transfer techniques suffice. However, some operators only support online signing while others offer non-standard DNS features. DNSSEC in this setting is challenging and does not yet have a suitable solution.

2.2 Threshold ECDSA

Threshold signatures enables a group of n parties to compute a signature only if a certain threshold, say $t + 1 \leq n$, participate in the signing protocol. Moreover, any group of t or less parties will neither learn anything about the signing key being used, nor will they be able to generate a valid signature on a previously unsigned message. The appeal of such protocols is that they make it possible to distribute the signing process and they make it more robust to adversarial attacks or disruptions, while preserving the original verification procedure of the signature scheme. In particular, signatures that are generated by the threshold protocol can be verified as if it were computed by a single party.

Protocols for computing signatures in distributed manner date back to the late 1990s and early 2000s, with protocols for DSS [35], RSA [55,24] or DSA [49]. Works on threshold signatures, specifically threshold ECDSA, has seen a resurgence. This renewed interest can largely be attributed to their role in crypto currency schemes where security of the signing key is paramount as it is used to authorize transactions; in a nutshell, the key *is* the account. Therefore, the possibility to increase the security of the key by splitting the role of the signer among several separate entities, without changing the verification procedure, makes it attractive to use threshold signatures in crypto currency schemes.

Multiple proposals for efficient threshold ECDSA protocols suitable for both 2 and n parties have been developed in the last couple of years. In the 2-party setting, Lindell [45] provides a protocol that achieves a throughput of over 100 signatures per second when using four threads. This protocol requires around 2435 ms to generate a key pair and 36.8 ms to sign using a single thread. Doerner et al. [27] propose a protocol that generates a key pair in 44.32 ms and computes a signature in just over 2 ms. However, the protocol of Doerner et al. relies heavily on Oblivious Transfer [52] that requires more communication than the protocol of Lindell, making it less ideal in low-bandwidth environments.

In the more general n-party setting, Gennaro et al. [34] present a protocol with a threshold of $t \leq n - 1$. A variant of their protocol that requires less communication is presented in [13]. Both these works require a distributed Paillier key generation scheme, which is inefficient. Gennaro and Goldfeder [33] present a protocol that does not require a dealer for key generation. [33] show that signing takes $29 + t \cdot 24$ ms in their work, $397 + t \cdot 91$ ms in [13] and $142 + t \cdot 52$ ms in [34]. Here t denotes the corruption threshold. [46,47] present a protocol which signs in 304 ms for $n = 2$ and in 3 seconds when $n = 10$. Key generation is significantly slower than signing and takes around 11 seconds for 2 parties and 17 seconds for 10 parties. Finally, a recent work by Doerner et al. [29] generalizes their 2-party protocol [27] to work for any number of parties. They achieve an impressive 37.6 ms for creating a signature with 24 parties and with $t = \lfloor (n-1)/2 \rfloor$. (Note that, as their protocol is dishonest majority, t dictates the number of *active* parties. In particular, 24 parties with $t = \lfloor (n-1)/2 \rfloor$ implies that only 13 parties actually participate in the protocol.)

3 System and Threat Model

Our system aims to take into account the diversity of the DNS ecosystem. Not only do we want to keep signing keys of domains secure, but we also want to make sure that the availability of the zone and, consequentially, zone signing is maintained under unexpected situations such as DDoS. In this section, we introduce the system and communication model we use in the rest of the work. We also introduce different threat models, along with the motivation behind each of them, that our solution supports.

3.1 System and Communication Model

Our system considers the scenario where the domain is served by more than one DNS operator. We assume that the operators can securely communicate with each other, e.g., using a TLS connection where both endpoints are authenticated. Furthermore, we assume that the domain owners can securely communicate with the DNS operators. Although a larger pool of operators increases the availability of the domain, domain owners also need to consider the cost-benefit trade-off. As two or three operators provides sufficient redundancy, we envision our system to be used in the setting where the number of DNS operators per domain is $n = 2$ or $n = 3$.

Our system focuses on the DNSSEC operations at the authoritative name servers. To be specific, we focus on the key generation and zone signing in the multi-operator setting. We assume that the domain owner is able to transfer the `DS` record to the registrar if the registrar is not one of the operators. Otherwise, our setting does not require the domain owner to be available for key generation and zone signing. In the setting we consider in this paper, no operator has access to the signing key. They only have access to the shares of the signing key. Hence, we eliminate the possibility of signing keys being leaked if a DNS operator is compromised.

3.2 Threat Model

As our distributed DNSSEC system may be used in different scenarios, we consider the guarantees that our system provides. We phrase these guarantees using standard MPC terminology as our solution supports all of them. First, we consider the guarantees that can be achieved when the threshold of honest parties varies. Second, we consider whether an adversary is malicious or not. In each case, we clarify the security that can be obtained.

Honest vs. dishonest majority. Let n be the number of DNS operators for a domain. In MPC, one kind of distinction that is made between different protocols is with regards to the number of parties that are needed to sign a message. A protocol wherein $t \leq \lfloor (n-1)/2 \rfloor$ are needed to sign is called *honest majority*, while the less restrictive case of $t < n$ is called dishonest majority. Honest majority protocols are typically faster and can be constructed to satisfy properties, such as guaranteed output, that dishonest majority protocols cannot achieve. Even if one of the parties has lost its key share, the rest of parties (as long as it is the majority of the parties, that is, two-out-of-three when $n = 3$) can run the protocol for zone signing. Thus, honest majority protocols are beneficial when availability of service is of prime importance. Nevertheless, there are situations when honest majority protocols do not suffice. Dishonest majority protocols provide stronger security than honest majority protocols since the adversary needs to corrupt all parties in order to access the signing key. On the other hand, this means that the service will become unavailable should one of the parties crash. Hence, there is trade-off between availability and security in these models.

Semi-honest vs. active security. Another way to classify MPC protocols is with respect to the adversary's capabilities. In a nutshell, either the adversary follows the protocol or it does not. An adversary of the former kind is called *semi-honest* or honest-but-curious, while the latter kind is called *active* or malicious. Several real life scenarios can be modelled by a semi-honest adversary. In a situation where the DNS operators trust each other but do not and cannot share signing keys of domains with each other due to their contract with the domain owner or due to legal reasons, a protocol constructed to be secure against semi-honest adversary

is sufficient. Such a protocol keeps private information away from the hands of an eavesdropping employee at the DNS operator.

We also consider an active adversary that has full control over the actions of one or more of the operators. In particular, it can act inconsistently or send wrong values during the protocol. It should not be surprising that security against semi-honest adversaries is easier to obtain than against active adversaries. Moreover, semi-honest protocols are typically much more efficient. In particular, since semi-honest adversaries are assumed to always supply the correct information during protocol execution, no extra checks are needed to ensure that computations are performed correctly.

4 Threshold ECDSA

A signature scheme consists of three operations—key generation, signature creation and signature verification—that are defined as follows:

- $\mathsf{KGen}(1^\lambda)$ on input a security parameter 1^λ, outputs a key pair $(\mathsf{sk}, \mathsf{pk})$ where sk is used to create signatures and is kept secret, and pk is used for verification and made public.
- $\mathsf{Sig}(\mathsf{sk}, M)$ on input the signing key sk and message $M \in \{0,1\}^*$, produces a signature σ.
- $\mathsf{Vf}(\mathsf{pk}, M, \sigma)$ on input the verification key pk, message M and signature σ, outputs 1 if σ is a valid signature on M and 0 otherwise.

If needed, we assume that participants agree on a set of public parameters beforehand, and that these parameters are provided as implicit input to all of the above functions.

4.1 ECDSA

For zone signing, the two most widely used signing algorithms are RSA and ECDSA. RSA continues to be widely deployed while the rate of adoption of ECDSA is increasing since it was standardised in 2012 [40,21,61]. As mentioned, our focus is on ECDSA which we introduce next. ECDSA as standardized in [43] is defined as follows: The scheme is parameterized by a subgroup $\mathcal{G} \subseteq E(K)$ of prime order p of some curve $E(K)$, where G is a generator of $\mathcal{G}$. We use $\mathbb{Z}_p$ to denote a field of order p and H to denote a hash function mapping messages unto elements of $\mathbb{Z}_p$.

- $\mathsf{KGen}(1^\lambda)$
  1. Sample at random $\mathsf{sk} \leftarrow \mathbb{Z}_p$ as the signing key.
  2. Compute $\mathsf{pk} = \mathsf{sk} \cdot G$ as the public verification key.
  3. Output $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{Sig}(\mathsf{sk}, M)$
  1. Sample an instance key $k \leftarrow \mathbb{Z}_p$ at random.
  2. Compute $(r_x, r_y) = k \cdot G$. If $r_x \equiv 0 \pmod p$, go back to Step 1.
  3. Compute $s = k^{-1}(H(M) + \mathsf{sk} \cdot r_x)$.
  4. Output $\sigma = (r_x, s)$.
- $\mathsf{Vf}(\mathsf{pk}, M, \sigma)$
  1. Parse σ as (r_x, s).
  2. Compute $(r'_x, r'_y) = s^{-1}(H(M) \cdot G + r_x \cdot \mathsf{pk})$.
  3. Output 1 iff $r'_x = r_x$.

It is easy to verify that Sig produces a valid signature:

$$\begin{aligned} & s^{-1}(H(M) \cdot G + r_x \cdot \mathsf{pk}) \\ & \quad = k(H(M) + \mathsf{sk} \cdot r_x)^{-1}(H(M) \cdot G + r_x \cdot \mathsf{pk}) \\ & \quad = k \cdot G \cdot ((H(M) + \mathsf{sk} \cdot r_x)^{-1}(H(M) + \mathsf{sk} \cdot r_x)) \\ & \quad = k \cdot G = (r_x, r_y). \end{aligned}$$

Security. Unlike its close relative, the Schnorr Signature scheme [54], it is not known whether ECDSA is secure even in the Random Oracle model. That being said, recent years have seen some work towards proving ECDSA secure in specifically tailored models, such as [32]. And if nothing else, the widespread use of ECDSA, nevertheless, provides strong empirical evidence towards its security.

4.2 Secure Multiparty Computation

We assume an MPC engine supporting the standard commands of the *arithmetic black-box (ABB)* functionality as shown in Figure 1, where the notation $[a]$ indicates that the value a is "secret-shared", i.e., that no party has access to it.

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 rerandomization technique [10].)
- 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. 1: The Arithmetic Black-Box Functionality.

4.3 Secure Computation on Arbitrary Groups

We present a novel 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 group used for doing MPC is irrelevant, as long as it is possible to perform linear operations. This generalization of arithmetic MPC has also been observed independently by [57], and might have applications in other contexts. In this paper, we use this idea to perform MPC in the subgroup $\mathcal{G}$. Importantly for our application, this extension comes at no extra cost in terms of communication and only a slight increase in complexity in terms of computation (corresponding to standard operations in the subgroup of the curve).

Consider a protocol implementing the ABB in Figure 1 and assume that the shares $[a]$ are also elements of $\mathbb{Z}_p$ (this is the case for most practical protocols based on additive, replicated, or Shamir's secret sharing). 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 an 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 $a \cdot G$.

The following two commands are added to the ABB:

- 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, along with the functionality of the original ABB (which provide secure computation over $\mathbb{Z}_p$) is enough 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 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 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 Figure 2.

Extended Arithmetic black-box (ABB+)

– $\mathsf{RandMul}$, $\mathsf{Mul}([\cdot], [\cdot])$, Rand, $\mathsf{Open}([\cdot])$ as described in Figure 1.
– 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. 2: ABB from Figure 1 extended to support computation over elliptic curves.

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. A natural question to ask is whether the active security guarantees of the $\mathbb{Z}_p$ protocol extend to the $\mathcal{G}$ protocol. We provide a positive answer to this question by showing that the MAC scheme of SPDZ [23] 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 [22]. Recall that 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:[6]

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.[7]

[6] Note that several openings can be batched at the same time, see the original paper for more details
[7] A partial opening entails revealing the value but not the MAC.

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. Suppose, in addition, the adversary lies about its MAC in Step 2 of SPDZ opening phase and let Δ denote this error. The adversary is successful if $\Delta = \sum_i \sigma_i$. In this case, we have

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

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 $\mathsf{cv}(a) = \mathsf{Convert}(a)$ 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

$$\begin{aligned}\langle a\rangle = ((&\mathsf{cv}(a_i), \ldots, \mathsf{cv}(a_N)),\\ &(\mathsf{cv}(\gamma(a)_i), \ldots, \mathsf{cv}(\gamma(a)_N))),\end{aligned}$$

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

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

We can therefore use the same $[\![\alpha]\!]$ to authenticate the cv'ed share as well. More precisely, we consider a modified opening procedure that works as follows:[8]

1. Let α_i be the share of the key held by P_i, and $\Gamma_i = \mathsf{cv}(\gamma(a)_i)$ be the shares of the MAC on $A = \mathsf{cv}(a)$.
2. Each P_i computes $\Sigma_i = \Gamma_i - \alpha_i A$ 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 in a straightforward manner, due to the linearity of the group operations, that 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 have information theoretic security, it is straight forward to reduce the security of the "MAC in the exponent" to the security of the regular MAC (as the reduction can run in unbounded time and retrieve the original MAC). Note that this would not have been the case with a MAC which was only computationally secure.

Other secret sharing schemes. The MAC check described in the previous section was particular to additive sharing, i.e., sharing a value a as $(a_1, \ldots, a_n)$ such that $a = \sum_i a_i$. However, it is worth pointing out that this is by no means a requirement. Indeed, all the arguments made in the previous section work as long as $\mathsf{Convert}$ preserves linearity, which is the case for the additive case (as shown), but also for Shamir secret sharing, replicated secret sharing, etc.

Active security using other approaches. We focus on SPDZ as that is what we benchmark in Section 5. However, the transform described so far also applies to a number of other transforms that we mention here in brief.

For example, the compiler of Chida et al. [17] achieves active security by keeping track of a randomized version of the value on a wire. I.e., each share is the pair $([a], [r \cdot a])$ where r is random and unknown to all

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

parties. At the end of the protocol parties compute a linear combination of the shares on the input wires and output wires, i.e.,

$$[u] = \sum_i \alpha_i \cdot [r \cdot z_i] + \sum_j \beta_j \cdot [r \cdot v_j]$$
$$[w] = \sum_i \alpha_i \cdot [z_i] + \sum_j \beta_j \cdot [v_j].$$

Next r is revealed and parties check if

$$0 \stackrel{?}{=} \mathsf{Open}([u] - r \cdot [w])$$

It is easy to see that if we define a $\mathcal{G}$ sharing as the pair obtained by applying $\mathsf{Convert}$ on $[a]$, i.e., $(\mathsf{Convert}([a]), \mathsf{Convert}([r \cdot a]))$ then the above check would still work and provide active security for computation over $\mathcal{G}$ as well.

4.5 Multiparty ECDSA protocol using the ABB+

We recall the protocol in [33] 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 it that does not reveal information about k. However, the inversion trick by Bar-Ilan and Beaver [9] 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 using $\mathsf{RandMul}$. Then, using $\mathsf{Convert}$, the parties can compute the value $R = \mathsf{Open}(\mathsf{Convert}([k]))$.

Recall that the other value we need is a sharing of sk/k. Given $[k^{-1}]$ it is straight forward to get $[\mathsf{sk}/k]$ 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 $R = k \cdot G$ 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 Figure 3.

The security of the full protocol is straightforward in the ABB+ hybrid model, i.e., the threshold ECDSA protocol will inherit the security properties of the underlying ABB+ (passive or active security, and corruption threshold).

Optimized SPDZ Opening. Threshold signatures are a very 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 been previously 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. This gives us a significant speedup in our SPDZ-based implementation since we can save the extra round of communication required for checking the correctness of the MAC, without any effect on the security. The only feasible attack against the optimistic opening in SPDZ is an additive attack, i.e., the attacker can make the honest parties output $s + \epsilon$ for an error value $\epsilon \neq 0$, which results in an invalid signature but does not disclose any information about the secret key.

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 $(R, [k^{-1}])$ 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 $[k] = [a]$ and compute $[k^{-1}] = c^{-1}[b]$.
4. Define $\langle k \rangle \leftarrow \mathsf{Convert}([k])$.
5. Run $R \leftarrow \mathsf{Open}(\langle k \rangle)$.
6. Output $(R, [k^{-1}])$.

User dependent preprocessing.

1. Take as input $[\mathsf{sk}_j]$ (the sharing of the secret key of user U_j) and $(R, [k^{-1}])$ (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 $(R, [k^{-1}], [\mathsf{sk}'_j])$.

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

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

Fig. 3: Protocol with preprocessing computing threshold ECDSA signatures using our extended ABB.

	AB (ms)	AC (ms)	BC (ms)
Colocated	0.07	0.07	0.08
Continent	11.51	16.98	8.2
World	240.0	71.0	181.1

Table 1: Round trip time between the servers used. A is always Ireland, while B is Paris and Seoul in the "Continent", respectively "World" setting, and C is London and N.Virginia in the "Continent", respectively "World" settings.

5 Evaluation

5.1 Implementation

We have implemented our protocol on top of MP-SPDZ [25] (one of the main frameworks for MPC for arithmetic circuits) and have used Crypto++ as the library for computation over elliptic curves.

MP-SPDZ provides several protocols for computation in $\mathbb{Z}_p$ with various security models. Of these, we have used *MASCOT*, *Semi* (i.e., semi-honest, dishonest majority), *Shamir* (malicious and semi-honest), and *Rep3* (i.e., three-party with replicated secret sharing, malicious and semi-honest). The first two protocols are based on oblivious transfer and work with a dishonest majority while the latter two use multiplicative secret sharing and work with an honest majority. Note that we have not used any protocol based on homomorphic encryption because the MP-SPDZ implementation of homomorphic encryption only supports the primes p which are incompatible with ECDSA, namely they must be such that 2^n divides $p-1$ for some n around 15.

5.2 Benchmarks

We have benchmarked our implementation with $n = 2$ and $n = 3$, depending on whether the underlying protocol requires an honest majority or not. In our benchmarking we split the protocol between preprocessing (including both preprocessing phases in Figure 3) and online signing phase. We measure both the throughput (based on batches of 10000 signatures), and the time for generating a single signature given a preprocessed tuple.

We used AWS `c5.2xlarge` instances in three settings: *Colocation*: All servers located in the same region. We used Ireland. *Continent*: Servers are located in the same continent. We used servers in Ireland, Paris and London. For two-party protocols, benchmarks are run between Ireland and Paris. *World*: Servers are located in different continents. We use a server in Ireland, one in North Virginia and one in Seoul. For two-party protocols, the servers in Ireland and Seoul were used. Round trip time (RTT) for our setup can be found in Table 1.

All benchmarks were run with a single thread. Our results, as well as comparisons with previous work, can be seen in Table 2 and Table 3. We explain our results first before we turn our attention to the comparisons with previous works. Notably (and unsurprisingly), the time required for one signature increases considerably with the distance between the parties because the protocols take up to two rounds of communication while the computation is rather straightforward. In particular, there are no elliptic curve operations because they only take place during the preprocessing.

The OT-based (MASCOT and Semi) and Rep3 protocols only take one round of communication for signing if the resulting s is checked instead of using the check in the protocol,[9] and this round clearly dominates the time. In theory, this would mean that time is roughly half the RTT. However, we found the results varying between RTT and virtually zero most likely because the parties did not act synchronously. Therefore, we decided to report the maximum over ten executions for the continent and the world setting, which comes down to the RTT. On the other hand, in the colocated setting, we use the checks offered by

[9] This does not apply for r_x, which is checked during preprocessing.

	Malicious	Full threshold	Tuples/s	Sig (ms)	KGen (ms)
MASCOT	✓	✓	291.56	0.22	3.84
Mal. Shamir	✓	✗	742.15	0.26	1.87
Mal. Rep3	✓	✗	927.61	0.26	1.31
Semi. OT	✗	✓	1428.15	0.12	1.15
Shamir	✗	✗	1095.19	0.25	1.23
Rep3	✗	✗	909.64	0.15	1.26
DKLS [27]	✓	✓	279.33	3.58	43.73
Unbound [59]	✓	✓	88.26	11.33	315.96
GG18† [33]	✓	✓	18.87	53.00	N/A
Lindel17† [45]	✓	✓	27.17	36.80	2435.00
LNR18† [47]	✓	✓	3.29	304.00	11 000.00

Table 2: LAN benchmarks for signing and key generation. Values in entries marked † are reported from the respective paper.

			Continent			World		
	Malicious	Full threshold	Tuples/s	Sig (ms)	KGen (ms)	Tuples/s	Sig (ms)	KGen (ms)
MASCOT	✓	✓	226.58	12.80	237.45	18.97	241.21	3031.35
Mal. Shamir	✓	✗	709.76	34.10	36.01	371.63	479.92	486.27
Mal. Rep3	✓	✗	890.17	18.70	41.35	577.27	240.64	433.50
Semi. OT	✗	✓	1202.33	11.20	67.44	201.42	230.48	935.15
Shamir	✗	✗	1052.99	20.35	35.12	564.80	484.22	486.13
Rep3	✗	✗	908.37	17.09	29.69	643.67	240.77	354.47
DKLS [27]	✓	✓	65.23	15.33	109.80	4.27	234.37	1002.97
Unbound [59]	✓	✓	32.18	31.08	424.02	2.04	490.73	1010.98

Table 3: WAN benchmarks for signing and key generation.

the protocol because it takes about 1 ms to check a signature, and we report averages over ten executions. For previous work, we always report averages.

It might be surprising that the semi-honest OT-based protocol has the highest throughput during preprocessing in the colocated and the same-continent settings when dishonest-majority protocols are usually more expensive than honest-majority protocols. This is explained by the fact that elliptic curve computation is by far the most costly part of the protocol, and the simple additive secret sharing in Semi is advantageous because only one share is converted from $\mathbb{Z}_p$. Replicated secret sharing uses two shares per value, and Shamir secret sharing requires relatively expensive operations of $\mathbb{Z}_p$ on the curve for reconstruction.

Table 4 shows the communication cost per party for the various protocols. As one would expect, dishonest-majority protocols are significantly more costly than honest-majority protocols, and malicious security is somewhat more costly than semi-honest security. Note that the minimal communication is exactly the size of one element of $\mathbb{Z}_p$. The figures for semi-honest Shamir are averages over the parties because the implementation is asymmetric.

Protocols from previous works. We benchmark against protocols from previous works where the implementation is publicly available (either as an artifact which was released with the paper or in the open-source codebase of companies): We ran the 2-party protocol from [27] using the code that we found at [28]. We also ran the protocol implemented by Unbound Tech, whose code we found at [59].[10] Note that, as is it usual

[10] This protocol is not (to the best of our knowledge) part of some published work. A white-paper describing this protocol can, however, be found in the repository (see *docs* directory).

	Preprocessing	Message-dependent	
		Protocol check	Offline check
MASCOT	1247.1	0.80	0.26
Semi. OT	198.0	0.26	0.26
Mal. Shamir	9.0	0.51	0.51
Semi. Shamir	2.9	0.34	0.34
Mal. Rep3	3.0	0.77	0.26
Semi. Rep3	1.4	0.26	0.26

Table 4: Communication per signature and party (kbit)

	Colocation		Continent		World	
	Secret (ms)	Public (ms)	Secret (ms)	Public (ms)	Secret (ms)	Public (ms)
MASCOT	1.82	2.02	201.83	35.62	2549.33	482.02
Semi. OT	0.59	0.56	49.65	17.79	692.28	242.87
Mal. Shamir	0.34	1.53	17.15	18.86	242.45	243.82
Shamir	0.19	1.04	17.10	18.02	242.50	243.63
Mal. Rep3	0.08	1.23	11.58	29.77	115.42	318.08
Rep3	0.08	1.18	11.55	18.14	111.29	243.18

Table 5: 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])).

the case, these comparisons are not "apples to apples" since we are comparing software written by different developers in different languages (but on the same network and computing configuration). However, it still provides an indication of the performances of the underlying protocols.

In addition, we also included some of the numbers reported in previous works (when it was not possible to replicate their experiments due to non-availability of the code): For the protocol in [33], we use the formula they provide with a threshold of 2. A similar approach is taken for the protocols of [45] and [47]. The authors of these works only consider a LAN setting, which is why we have chosen to only include their numbers in our LAN table. Finally, we chose not to include numbers from [29], as this protocol only becomes interesting for a large number of parties. For $n = 2$ or $n = 3$ (which is the number of parties we are concerned with), this protocol is just a slightly slower variant of [27].

Comparing our protocol to previous work. A direct comparison between our work and existing protocols from academy or industry is not possible. Indeed, contrary to existing protocols, ours relies on preprocessing to achieve the efficient signing.[11] However, in Table 2 and Table 3, we provide numbers indicating the number of signatures each protocol can generate per second. For existing protocols, these numbers were computed as $1000/x$ where x is the value in the Sig column, while for ours the column simply denotes the number of tuples computed per second.

Key Generation. Finally, we also run benchmarks for key generation. For the colocated setting, we report the average, while for the continent and world settings, the maximum is used. We note that the MASCOT protocol uses a less than optimal way of generating the shared secret key [sk] (see also Table 5, which shows the timings for both public and secret key generation). All our protocols are 10x faster than [27] and almost 100x faster than [59] in the colocated setting. Interesting to note is the seemingly huge benefit we get by switching to an honest majority protocol. In the world setting, our honest majority protocols are at least twice as [27] and [59].

[11] This is not to say that existing protocols could not be adapted to work in the preprocessing model e.g., the protocols of Doerner et al. [27],[29], which are based on OT extension, could probably be adapted to this model.

6 Measurements

Since the large DDoS attacks on Dyn [30] or NS1 [11] 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.

6.1 Approach

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 make use of 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 the 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 [2] as the dataset. The list was downloaded on 12th 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 SLDs* (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`. We do not only compare the second-level domain (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 [48]. 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 [19] and rely on the `NS` records to indicate the DNS operator. We made manual 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.

6.2 Results

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 second 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 Figure 4 for the result of classifying the Alexa Global Top 1 million, as well as its subsets, in this manner.

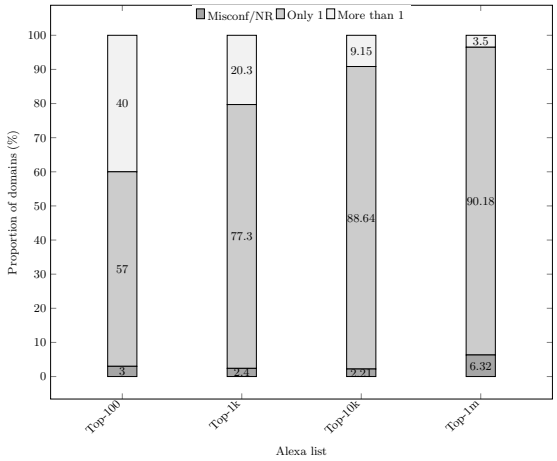

Fig. 4: Ratio of domains with multiple operators

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.

7 Multiparty zone signing system

We integrate MP-SPDZ with DNS administrative name servers. For DNS name server software, we used Knot DNS [44] 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.

7.1 Setup

Figure 5 shows the architecture of the proposed DNSSEC signing system at a DNS operator. Each operator serves a name server, runs a threshold ECDSA module and has two keys stores: one to store the keys for particular zones and another to store the key material associated with other operators. We do not change the operation of Knot DNS apart from the parts involved in DNSSEC key generation, key rollover and zone signing. All communication between the name server and the threshold ECDSA module is performed using a message queue, which takes on average 31 ms on our non-optimized proof-of-concept implementation. This timing is independent of the key generation/rollover and zone signing timings presented in Section 5.2.

We consider three name servers operated by independent DNS operators, all of which support ECDSA with SHA256 message digest. There is an intersection in the domains that they serve as their customers care about the availability of the name servers. However, either they do not trust each other to share signing keys for the zones they operate, or they are legally obliged to not share the signing keys. Hence, they resort to using threshold ECDSA protocol.

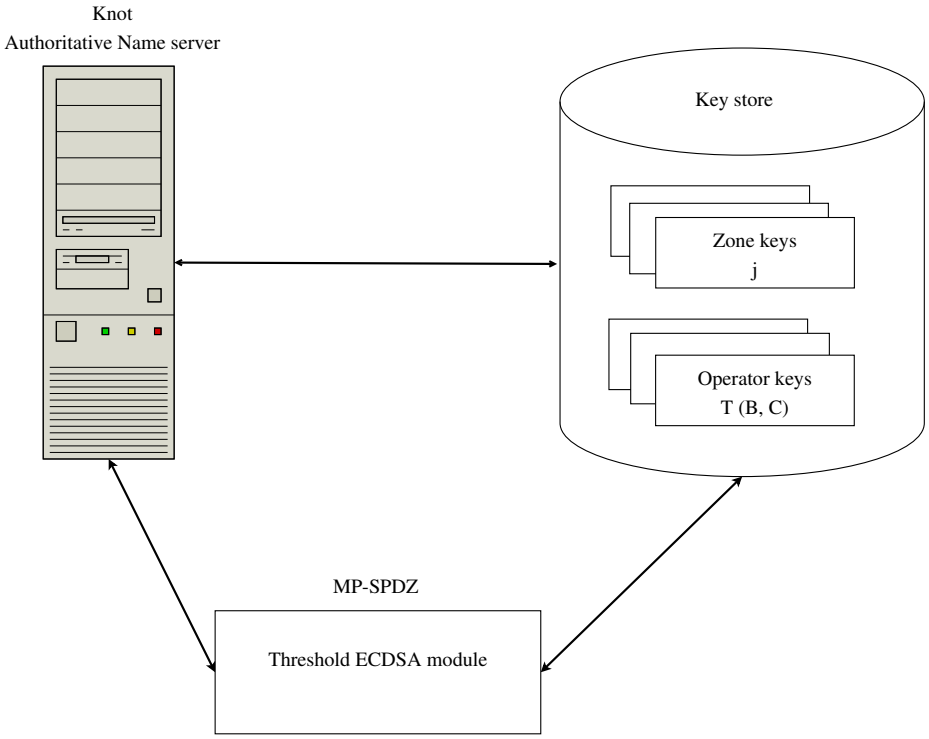

Fig. 5: Setup at DNS operator

7.2 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 Figure 3. 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. This is the only phase where the domain owner is involved.

7.3 Zone signing

As shown in Figure 3, our signing protocol has three processes: 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. We show the three processes and the steps involved in Figure 6.

User independent preprocessing is run between the threshold ECDSA modules to generate a bunch of tuples. Only the knowledge of the associated DNS operators needs to be known. Lets assume that Operator A initiates the zone signing process. In Step 1, the threshold ECDSA module receives $T(B, C)$ along with the request to generate tuples from the name server. In Step 2, the tuples $(R, [k^{-1}])$ are generated using the MPC protocol. In Step 3 (3p in Figure 6), the tuples are stored in the keystore for operator keys.

User dependent preprocessing is run between the threshold ECDSA modules to generate a bunch of signing key shares. In Step 1, For a particular zone j, Operator A sends $T(B, C)$ and $(R, [k^{-1}])$, $[\mathsf{sk}_j]$ from

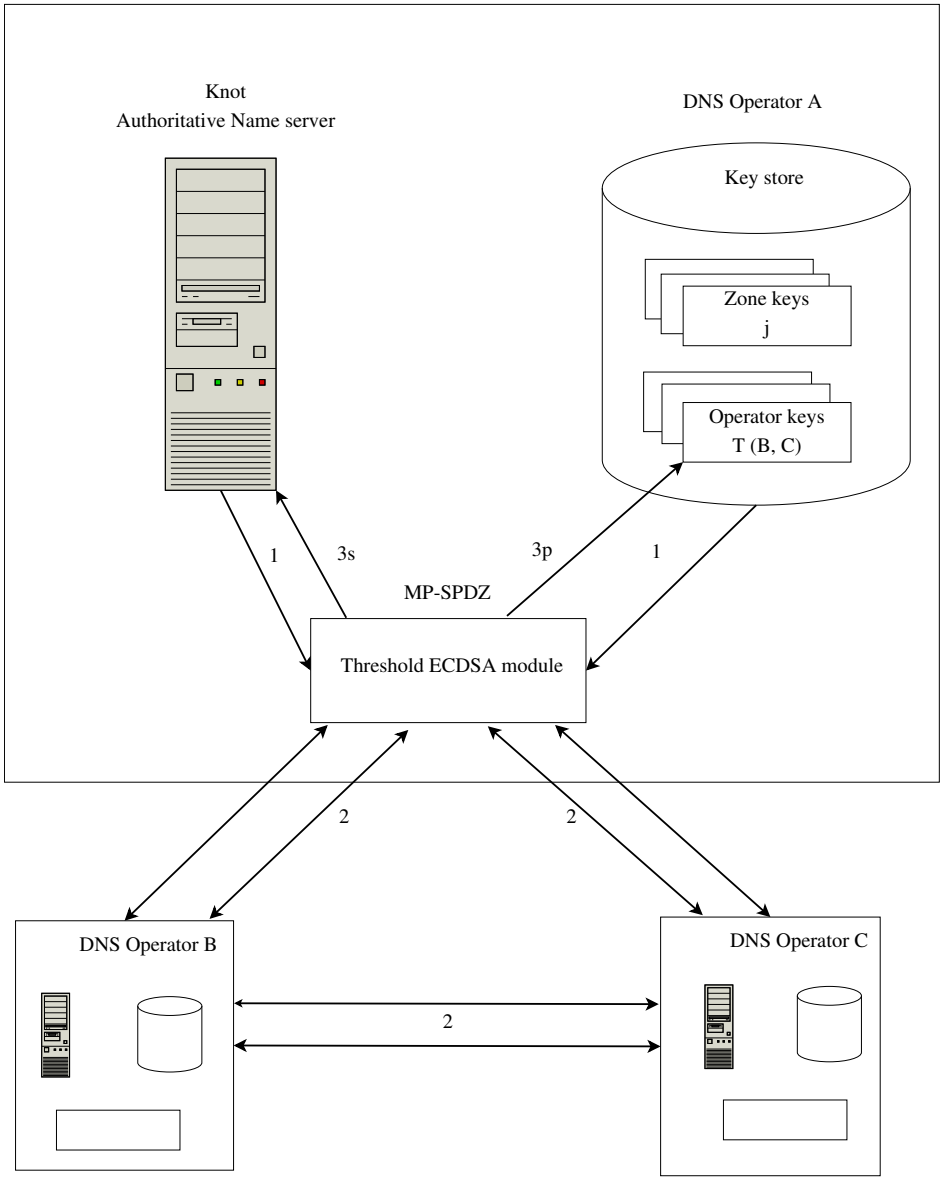

Fig. 6: Zone signing

the key store to the threshold ECDSA module. In Step 2, the signing key shares $(R, [k^{-1}], [\mathsf{sk}'_j])$ are generated using the MPC protocol. In Step 3 (3p in Figure 6), the signing key shares are stored in the key store. This step can be run at any time when sufficient signing key shares are not available.

When Operator A wants to sign a RRset in zone j, it sends the RRset along with the signing key share $(R, [k^{-1}], [\mathsf{sk}'_j])$ and the associated tag $T(B, C)$ to the threshold ECDSA signing module in Step 1. The threshold ECDSA module runs in the background and periodically polls the name server so that it is always available to sign. In Step 2, MPC signing protocol is run between the three DNS operators to generate the signature `RRSIG` for the RRset. In Step 3 (3s in Figure 6), the `RRSIG` is sent to the name server to store in the zone file.

7.4 At DNS resolvers

Proper functioning of the DNSSEC ecosystem requires both the signing part and the validation part to work. Deploying changes at DNS resolvers is extremely hard as numerous resolver software needs to be changed. Fortunately, no change is required at the validating resolver to use of 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 who 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 which operator responded to the query.

8 Related Works

Privacy in DNS. Though DNSSEC provides data integrity, it does not provide confidentiality. In [62] "range queries" are proposed to be used to hide a query within a set of dummy queries while in [63], private information retrieval (PIR) [18] is proposed as a solution to hide queries. However, a privacy analysis of the proposal of [62] by [36] concluded that in the scenario of web-surfing, inter-related DNS queries are issued by the client and this relation provides much more information than single queries. Another approach, PageDNS [7] lets TLD registries group together the name server records in their zones into pages; recursive resolvers retrieve entire pages rather than single records, which provides a first level of privacy protection. PageDNS makes extensive use of caches so that SLDs are not queried too often. They provide an analysis to show that SLD name servers are not updated very often and hence, one does not need to query them regularly. Finally, the Internet Engineering Task Force (IETF) has recently begun considering privacy issues in DNS and DNSSEC [14,15] and proposed DNS-over-TLS [26] and DNS-over-HTTPS [39].

DNSSEC deployment and measurement. DNSSEC deployment heavily relies on DNS operators and registrars. A few DNS operators reuse their keys for multiple domains that they manage [19]. It was shown in [56] that a large fraction of domains are signed using RSA keys that share the moduli with other domains. Cloudflare, for instance, advocates to use *a single global KSK and ZSK with multiple names*, i.e., one key for multiple domains as they believe they anyway need to change all the keys at the same time.[12] They claim that there is no reason to have millions of ZSKs and KSKs as the keys are used/stored/rolled together. Additionally, a large fraction of domains sign their `DNSKEY` record twice: once with the KSK (as expected) and once with the ZSK (which is not used in validation) [19]. This not only increases the `DNSKEY` packets sizes, but could lead to fragmentation attacks (if strong RSA keys of size 2048 bits or more are used). This makes domain resolution inefficient [60], but also makes DNSSEC vulnerable to poisoning attacks when resolvers do not validate responses [37]. With regards to measurements on the use of multiple operators, [1] measures the impact of DDoS attacks and how many customers of DyN and NS1 added another operator after the

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

DDoS attack of 2016. However, they only measure the domains that use DyN and NS1. In our work, we measure the use of multiple operators, not restricting our measurements to managed DNS providers.

A recent IETF draft [41] proposed deployment models in the multi-operator setting.[13] The deployment models presented include the use of either common KSKs among operators per zone or unique KSK and ZSK per operator per zone. The domain owner's participation is required not only to update the `DS` record, but also to update the `DNSKEY` record across the operators. Unless the same `DNSKEY` record is present at all operators, signature verification could fail at the resolvers when a `DNSKEY` of one operator is present in the cache and it is used to validate a response from another operator. Furthermore, in the case that unique keys are used, the operators remain in control of the keys. They also remark that the use of a shared KSK and ZSK scenario is not desirable (from an industry perspective) as the operators have a contract with domain owner and they do not want to share signing keys with other operators. We recognize this situation as one where our multi-party ECDSA can be applied.

Threshold signatures for DNSSEC. Threshold RSA signatures for DNSSEC have been considered in the past. [16] proposed a distributed DNS to avoid single point of failure, which provides fault tolerance and security in the presence of corrupted servers. They used RSA threshold signature scheme of [55] in their work. [20] also used [55] to emulate a HSM at an authoritative name server. They report timings on a LAN which range from tens to hundreds of milliseconds on commodity hardware.

9 Conclusion

We described 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 order p subgroup of an elliptic curve. 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. Next we performed a series of measurements to study the extent with which multi-operator solutions for name servers are currently used on the internet. These measurements reveal that a significant portion of domains utilize multiple distinct operators. Finally, motivated by the aforementioned measurements, we show that our protocols provide a very 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.

References

1. Abhishta Abhishta, Roland van Rijswijk-Deij, and Lambert J. M. Nieuwenhuis. Measuring the impact of a successful ddos attack on the customer behaviour of managed DNS service providers. *Computer Communication Review*, 48(5):70–76, 2018.
2. Alexa. Top 1m sites, July 12 2019. https://www.alexa.com/topsites http://s3.amazonaws.com/alexa-static/top-1m.csv.zip.
3. Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. DNS security introduction and requirements. *RFC*, 4033:1–21, 2005.
4. Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. Protocol modifications for the DNS security extensions. *RFC*, 4035:1–53, 2005.
5. Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. Resource records for the DNS security extensions. *RFC*, 4034:1–29, 2005.
6. Suranjith Ariyapperuma and Chris J. Mitchell. Security vulnerabilities in DNS and DNSSEC. In *ARES*, pages 335–342. IEEE Computer Society, 2007.
7. Daniele Asoni, Samuel Hitz, and Adrian Perrig. A Paged Domain Name System for Query Privacy. In *International Conference on Cryptology and Network Security (CANS)*, November 2017.
8. Derek Atkins and Rob Austein. Threat analysis of the domain name system (DNS). *RFC*, 3833:1–16, 2004.

[13] They use the term Multi provider to refer to the multi-operator setting.

9. Judit Bar-Ilan and Donald Beaver. 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*, pages 201–209. ACM, 1989.
10. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 420–432, 1991.
11. Kris Beevers. A note from ns1's ceo: How we responded to last week's major, multi-faceted ddos attacks, May 23 2016. https://ns1.com/blog/how-we-responded-to-last-weeks-major-multi-faceted-ddos-attacks.
12. Steven M. Bellovin. Using the domain name system for system break-ins. In *USENIX Security Symposium*. USENIX Association, 1995.
13. Dan Boneh, Rosario Gennaro, and Steven Goldfeder. Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security. 2017. http://www.cs.haifa.ac.il/~orrd/LC17/paper72.pdf.
14. Stephane Bortzmeyer. DNS privacy considerations. *RFC*, 7626:1–17, 2015.
15. Stephane Bortzmeyer. DNS query name minimisation to improve privacy. *RFC*, 7816:1–11, 2016.
16. Christian Cachin and Asad Samar. Secure distributed dns. In *International Conference on Dependable Systems and Networks, 2004*, pages 423–432. IEEE, 2004.
17. Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, pages 34–64, 2018.
18. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 41–50. IEEE, 1995.
19. Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. A longitudinal, end-to-end view of the DNSSEC ecosystem. In *USENIX Security Symposium*, pages 1307–1322. USENIX Association, 2017.
20. Francisco Cifuentes, Alejandro Hevia, Francisco Montoto, Tomás Barros, Victor Ramiro, and Javier Bustos-Jiménez. Poor man's hardware security module (pmhsm): A threshold cryptographic backend for DNSSEC. In *LANC*, pages 59–64. ACM, 2016.
21. Tianxiang Dai, Haya Shulman, and Michael Waidner. DNSSEC misconfigurations in popular domains. In *CANS*, volume 10052 of *Lecture Notes in Computer Science*, pages 651–660, 2016.
22. Ivan Damgard, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. 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.
23. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.
24. Ivan Damgård and Maciej Koprowski. Practical threshold rsa signatures without a trusted dealer. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, pages 152–165, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
25. Data61. MP-SPDZ - Versatile framework for multi-party computation. https://github.com/data61/MP-SPDZ.
26. Sara Dickinson, Daniel Kahn Gillmor, and Tirumaleswar Reddy. Usage profiles for DNS over TLS and DNS over DTLS. *RFC*, 8310:1–27, 2018.
27. J. Doerner, Y. Kondi, E. Lee, and A. Shelat. Secure two-party threshold ecdsa from ecdsa assumptions. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 980–997, May 2018.
28. Jack Doerner. mpecdsa. https://gitlab.com/neucrypt/mpecdsa/.
29. Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ecdsa from ecdsa assumptions: The multiparty case. page 0. IEEE.
30. DYN. Dyn analysis summary of friday october 21 attack, October 26 2016. https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.
31. Robert Elz, Randy Bush, Scott O. Bradner, and Michael A. Patton. Selection and operation of secondary DNS servers. *RFC*, 2182:1–11, 1997.
32. Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the provable security of (ec)dsa signatures. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1651–1662, New York, NY, USA, 2016. ACM.
33. Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *ACM Conference on Computer and Communications Security*, pages 1179–1194. ACM, 2018.

34. Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, pages 156–174, 2016.
35. Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold dss signatures. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 354–371, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
36. Dominik Herrmann, Max Maaß, and Hannes Federrath. Evaluating the security of a DNS query obfuscation scheme for private web surfing. In *SEC*, volume 428 of *IFIP Advances in Information and Communication Technology*, pages 205–219. Springer, 2014.
37. Amir Herzberg and Haya Shulman. Fragmentation considered poisonous, or: One-domain-to-rule-them-all.org. In *CNS*, pages 224–232. IEEE, 2013.
38. Amir Herzberg and Haya Shulman. Socket overloading for fun and cache-poisoning. In *ACSAC*, pages 189–198. ACM, 2013.
39. Paul E. Hoffman and Patrick McManus. DNS queries over HTTPS (doh). *RFC*, 8484:1–21, 2018.
40. Paul E. Hoffman and Wouter C. A. Wijngaards. Elliptic curve digital signature algorithm (DSA) for DNSSEC. *RFC*, 6605:1–8, 2012.
41. Shumon Huque, Pallavi Aras, John Dickinson, Jan Vcělák, and David Blacka. Multi Provider DNSSEC models. Internet-Draft draft-ietf-dnsop-multi-provider-dnssec-03, Internet Engineering Task Force, March 2019. Work in Progress.
42. Dan Kaminsky. Black ops 2008: It's the end of the cache as we know it. *Black Hat USA*, 2008.
43. Cameron F. Kerry and Patrick D. Gallagher. Fips pub 186-4 federal information processing standards publication digital signature standard (dss), 2013.
44. Knot. Knot DNS. https://www.knot-dns.cz/.
45. Yehuda Lindell. Fast secure two-party ECDSA signing. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 613–644, 2017.
46. Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1837–1854, 2018.
47. Yehuda Lindell, Ariel Nof, and Samuel Ranellucci. 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.
48. Suqi Liu, Ian D. Foster, Stefan Savage, Geoffrey M. Voelker, and Lawrence K. Saul. Who is .com?: Learning to parse WHOIS records. In *Internet Measurement Conference*, pages 369–380. ACM, 2015.
49. Philip MacKenzie and Michael K. Reiter. Two-party generation of dsa signatures. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 137–154, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
50. Paul V. Mockapetris. Domain names - concepts and facilities. *RFC*, 1034:1–55, 1987.
51. Paul V. Mockapetris. Domain names - implementation and specification. *RFC*, 1035:1–55, 1987.
52. Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.
53. Benjamin Rothenberger, Daniele Enrico Asoni, David Barrera, and Adrian Perrig. Internet kill switches demystified. In *EUROSEC*, pages 5:1–5:6. ACM, 2017.
54. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.
55. Victor Shoup. Practical threshold signatures. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, pages 207–220, 2000.
56. Haya Shulman and Michael Waidner. One key to sign them all considered vulnerable: Evaluation of DNSSEC in the internet. In *NSDI*, pages 131–144. USENIX Association, 2017.
57. Nigel P. Smart and Younes Talibi Alaoui. Distributing any elliptic curve based protocol: With an application to mixnets. Cryptology ePrint Archive, Report 2019/768, 2019. https://eprint.iacr.org/2019/768.
58. Sooel Son and Vitaly Shmatikov. The hitchhiker's guide to DNS cache poisoning. In *SecureComm*, volume 50 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 466–483. Springer, 2010.
59. Unbound Tech. blockchain-crypto-mpc. https://github.com/unbound-tech/blockchain-crypto-mpc.
60. Gijs Van Den Broek, Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. Dnssec meets real world: dealing with unreachability caused by fragmentation. *IEEE communications magazine*, 52(4):154–160, 2014.

61. Roland van Rijswijk-Deij, Mattijs Jonker, and Anna Sperotto. On the adoption of the elliptic curve digital signature algorithm (ECDSA) in DNSSEC. In *CNSM*, pages 258–262. IEEE, 2016.
62. Fangming Zhao, Yoshiaki Hori, and Kouichi Sakurai. Analysis of privacy disclosure in DNS query. In *MUE*, pages 952–957. IEEE Computer Society, 2007.
63. Fangming Zhao, Yoshiaki Hori, and Kouichi Sakurai. Two-servers PIR based DNS query scheme with privacy-preserving. In *IPC*, pages 299–302. IEEE Computer Society, 2007.