DistriTrust: Distributed and Low-Latency Access Validation in Zero-Trust Architecture[☆]

Binanda Sengupta[*], Anantharaman Lakshminarayanan

Institute for Infocomm Research
*Agency for Science, Technology and Research (A*STAR), Singapore*

Abstract

Enterprise networks typically use perimeter-based security to protect their IT resources from security threats originating from outside the enterprise perimeter. In such networks, connection requests from devices residing inside the perimeter are implicitly assumed to be trusted. However, in practice, a considerable fraction of cyberattacks emanate from inside the enterprise network making the perimeter-based trust assumption unrealistic. *Zero-trust architecture* (ZTA) is an emerging alternative to perimeter-based security, where each connection request for accessing an enterprise resource goes through stringent security checks and validation irrespective of the location of the requesting device. In ZTA, every connection request is authenticated and authorized by a trusted centralized component called the *policy decision point* (PDP) and subsequently granted (or denied) access to the requested resource. However, the centralized nature of the PDP often makes it vulnerable to various attacks such as compromise of secret keys, impersonation and denial of service. In this article, we propose `DistriTrust` which distributes trust across multiple PDPs using the notion of threshold signatures. However, involving multiple PDPs also increases latency. In order to keep latency as low as possible, we study different threshold signature schemes and identify a suitable scheme for `DistriTrust`. We also discuss the security properties achieved by `DistriTrust`. Finally, we analyze the asymptotic performance of `DistriTrust` and report the experimental results as well.

Keywords: Cybersecurity, zero-trust architecture, access validation, threshold signatures, low latency.

[☆]This is the accepted version of the article published in the Journal of Information Security and Applications (Elsevier) (the final published version is available at ScienceDirect: https://doi.org/10.1016/j.jisa.2021.103023).

[*]Corresponding author.

Email addresses: `binandas@i2r.a-star.edu.sg` (Binanda Sengupta), `lux@i2r.a-star.edu.sg` (Anantharaman Lakshminarayanan)

1. Introduction

Perimeter-based network security has been widely used in enterprises in order to secure access to enterprise resources [1]. These resources include critical data (e.g., intellectual property, payment information, personally identifiable information, protected health information), software applications, assets (e.g., SCADA controls, manufacturing assets, Internet-of-Things or IoT devices, servers for computation/storage, medical equipments) and services (e.g., DNS, DHCP, Active Directory) [2]. When perimeter-based network security is deployed, the devices within an enterprise network perimeter are assumed to be trusted, whereas access requests coming from devices outside the perimeter, e.g., through VPN, are checked for possible cyber threats. However, the assumption on every user/device within the perimeter being trustworthy is not often realistic [3, 4]. Thus, a malicious user/device, which is somehow able to get inside the perimeter once, can move freely within the perimeter and access certain restricted enterprise resources. Moreover, with the advent of 5G and massive machine-type communications involving thousands of devices connected to enterprise networks via the Internet, many enterprises are outsourcing their resources and services to remote clouds for efficiency and ease of maintenance [5] — which essentially extends an enterprise network beyond its physical perimeter.

Zero-trust architecture (ZTA) [6] has been proposed as a more secure alternative to perimeter-based security solutions for enterprise networks. In ZTA, stringent security checks and validation are performed irrespective of the location of the requesting device. Due to numerous incidents regarding insider breaches and cyberattacks reported in recent years [7, 3], ZTA is gaining popularity among major enterprises (e.g., Google has deployed its zero-trust network called BeyondCorp [8, 9]; Microsoft is also implementing its own zero-trust security model [10, 11]). In view of such substantial research interest vested in ZTA, the National Institute of Standards and Technology (NIST) has very recently published a draft on ZTA in order to develop the terms, definitions, and components of ZTA network infrastructure [12].

In ZTA, when a user/device wants to access an enterprise resource, the connection request is initiated and validated by a *policy enforcement point* (PEP) and a *policy decision point* (PDP) [12]. The entities involved in setting up a connection in a zero-trust system are shown in Figure 1. The PEP is split into two components: an *agent* (user side) and a *gateway* (resource side). The agent requests the PDP for a new connection to access the resource, and the PDP validates the request and approves/rejects it accordingly. Finally, the resource gateway establishes a connection between the user/device and the requested resource if and only if the connection request is approved by the PDP.

Problem Statement. In ZTA, the policy decision point (PDP) decides if a request coming from a user/device to access an enterprise resource can be granted. In other words, the connection can be established only if the PDP approves the same. Thus, the PDP is a crucial component in ZTA which can be considered as a point that every connection request must go through. Two issues

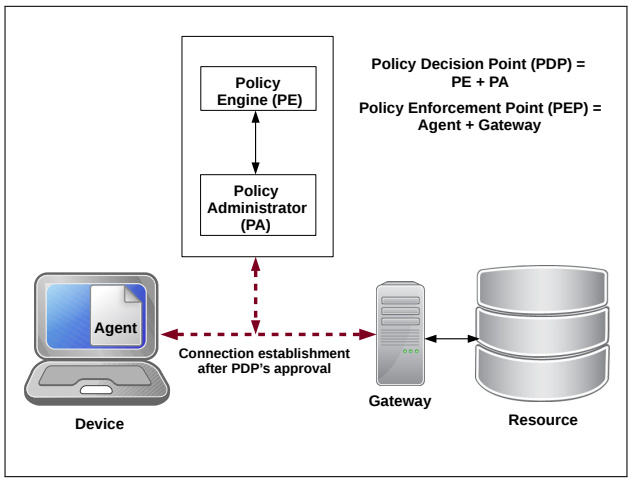

Figure 1: Entities involved in access validation of a connection request in a zero-trust enterprise network.

ensue: 1) the PDP becomes a single point of failure (e.g., the network collapses in case the PDP goes offline); 2) the PDP becomes a single point of trust that an attacker can attempt to subvert (e.g., invalid requests are granted in case the PDP is compromised) — which necessitates the elimination of dependency on a single PDP in order to make a zero-trust network more robust and fault-tolerant.

Involving multiple PDPs for robust access validation incurs an overhead in latency due to increased communications among different entities in a network. However, as numerous connection requests are made by various users/devices for different enterprise resources, the latency overhead incurred due to distributed validation must be kept as low as possible — which requires the access validation in a zero-trust network to be robust as well as efficient.

Our contribution. We summarize our major contributions towards designing distributed and low-latency access validation in ZTA as follows.

- We propose a solution that addresses the aforementioned problem by employing multiple PDPs, thereby distributing trust among these PDPs instead of relying on a single PDP. In our solution, we exploit techniques from threshold signatures in order to achieve resilience against single point of failure/corruption. This is done by splitting the corresponding secret signing key into key-shares and distributing these key-shares among the PDPs, such that the signing key is not available to any single PDP. To be precise, in order to access an enterprise resource, a connection request made by a user/device must be approved (or signed) by at least a predetermined number (say, k) of PDPs using threshold signatures. On the other hand, any set of dishonest PDPs with cardinality less than k cannot produce a valid (threshold) signature for an invalid or unauthorized connection request. To the best of our knowledge, ours is the first work that

increases resilience of access validation in ZTA. In order to compromise the validation process, an attacker now has to break into a multiplicity of PDPs that are possibly distributed across various geographical locations.

- As a large number of connection requests are made by different devices across an enterprise network for accessing its resources, each of these requests must be served efficiently and in real time — which requires the latency involved in access validation to be low. We thoroughly analyze different threshold signature schemes based on their latency (both the computational cost and the number of communication rounds required) and identify a threshold signature scheme, among these candidates, that appears to be most suitable in the context of ZTA. Our solution, that we call `DistriTrust`, for authorizing valid connection requests in ZTA exploits this threshold signature scheme.
- We describe the threat model for multiple PDPs validating access requests in ZTA and discuss how `DistriTrust` achieves certain security properties – such as authenticity, authorization and confidentiality – required for a zero-trust enterprise network.
- Finally, we analyze the asymptotic performance of `DistriTrust` based on the computational and communication costs it incurs. We implement a prototype of `DistriTrust` and report the experimental results in order to show its efficiency. We also discuss how the threshold techniques of `DistriTrust` can be applied to OAuth 2.0, a standard authorization framework, in order to authorize a connection collaboratively in a more robust way.

Organization. The rest of this article is organized as follows. In Section 2, we discuss certain preliminaries and background related to our work. Section 3 describes the system model including underlying assumptions and threat model. In Section 4, we describe `DistriTrust`, a low-latency solution for distributed access validation in ZTA. Section 5 discusses the correctness and security of `DistriTrust`. We analyze the performance of `DistriTrust` in Section 6. In Section 7, we discuss certain architectural and operational elements supporting `DistriTrust` and certain risk metrics associated with the number of PDPs in `DistriTrust`. Section 8 discusses how the threshold techniques of `DistriTrust` can be applied to OAuth 2.0. Finally, we conclude the article in Section 9.

2. Preliminaries and Background

In this section, we describe the components in ZTA which are responsible for access validation. We note that our solution `DistriTrust` distributes access validation in ZTA using threshold signatures. We discuss some of the existing threshold signature schemes found in the literature. Finally, we briefly mention some notations and a number-theoretic function called Jacobi symbol that we use later in this article.

2.1. Access Validation in Zero-Trust Architecture

Zero-trust architecture (ZTA) was proposed by John Kindervag of Forrester Research as an alternative to legacy perimeter-based security infrastructure for enterprise networks [6]. For perimeter-based security, the users/devices residing within an enterprise network perimeter are assumed to be trusted and security checks are performed only if connection requests come from outside the perimeter. In contrast, ZTA considers trust as a vulnerability, and checks for authentication and authorization of each request to access enterprise resources – such as data sources, applications, computing/storage servers – irrespective of the physical location of the requesting users/devices.

A connection between a user/device and an enterprise resource in a zero-trust network is approved and established through a policy decision point (PDP) and a policy enforcement point (PEP) [12]. As shown in Figure 1, the PDP consists of a policy engine (PE) and a policy administrator (PA), and the PEP consists of an agent (installed at the respective device) and a gateway (placed in front of the requested resource). The local agent sends a connection request to the PA which forwards the request to the PE for validation. In order to decide the validity of a connection request, the PE runs a *trust algorithm* which takes the following as input [12].

- *Connection request*: A connection request includes the information of the user (e.g., user identities required for authentication such as account ID/password, biometric data like fingerprints, facial recognition data), the information of the device (e.g., device identity such as device certificate, network location of the device, information of the application that requires the access), the information of the enterprise resource to access (e.g., IP address).

- *User database*: An enterprise maintains a user database that contains the details of every user in the enterprise. This database includes the names of all valid users, their authentication information, roles, the lists of enterprise devices they own, and other relevant information users need for accessing enterprise resources. This database is continuously monitored, analyzed and updated by the network administrator of the enterprise.

- *System database*: An enterprise maintains a system database that contains the details of each of the devices (e.g., MAC address, OS version, patch level, device certificates stored in a hardware or software trusted platform module) managed by the enterprise. Similar to the user database, the system database is also continuously monitored, analyzed and updated by the network administrator.

- *Resource access policies*: An access policy for the requested resource defines the minimal requirements to be satisfied for accessing the resource. For example, it may include an assurance level of the authentication procedure like multi-factor authentication (MFA) and network location of the device (e.g., an enterprise may not entertain connection requests coming from overseas IP addresses).

- *Threat intelligence services*: Information about general threats recently developed on the Internet and the ways they can be mitigated are fed as input to the trust algorithm. Enterprises may opt for such services from third parties in order to protect their networks against such threats.

Based on the inputs mentioned above, the trust algorithm decides whether to approve/reject the access request for the respective resource. The resource gateway establishes a connection between the user/device and the requested resource if and only if the connection request is approved by the PDP.

2.2. Threshold Signatures

Given parameters k and l with $l \geq k$, a k-out-of-l threshold signature scheme enables any k-subset (out of total l players participating in the protocol) to generate a signature collaboratively. From the security point of view, threshold signature schemes must satisfy a property called *unforgeability*: any subset of t malicious participants (such that $t < k$) cannot produce a valid signature even if they collude with each other. A threshold signature scheme is called *robust* if the corrupted participants cannot prevent k or more uncorrupted participants from generating valid signatures. In a typical k-out-of-l threshold signature scheme, each participant possesses a share of the secret signing key which is used to produce a share of a signature on a given message. When signature-shares are obtained from any k uncorrupted participants, these shares can be combined to produce a final signature on the same message. The verification procedure is same as that in the standard (i.e., non-threshold) version of the underlying signature scheme.

Based on the concept of group-oriented cryptosystem [13, 14] for securing the communications among the members of a group, Desmedt and Frankel [15] construct a threshold signature scheme using Rivest-Shamir-Adleman (RSA) signatures [16] and Shamir's secret sharing [17]. Following their scheme, several threshold signature schemes are proposed based on either assumptions related to integer factorization (e.g., threshold signature schemes based on RSA) or assumptions related to discrete logarithm (e.g., threshold signature schemes based on Digital Signature Standard or DSS, Digital Signature Algorithm or DSA, Elliptic-Curve DSA or ECDSA). Some of these schemes assume the existence of a trusted dealer which generates and distributes the shares of the secret key to the respective participants [18, 19, 20, 21], whereas other schemes do not rely on this assumption.

Gennaro et al. [18] propose an efficient and robust $l/2$-out-of-l threshold RSA signature scheme where the robustness property is achieved by employing extensions of the information checking protocol of [22, 23] and an undeniable signature [24]. Rabin [19] proposes a threshold RSA signature scheme where the secret key is shared using an l-out-of-l additive sharing among the l participants. Each of these shares is further shared using a verifiable secret sharing scheme [25]. Miyazaki et al. [26] build an RSA-based threshold signature scheme with no trusted dealer. Unlike the existing RSA-based schemes, the modulus

used in their scheme is not a product of two safe primes.[1] Shoup [21] proposes an efficient and robust k-out-of-l (for $k \geq t+1$) threshold signature scheme based on the RSA signature scheme which can withstand the participation of malicious participants providing incorrect signature-shares. During the combination phase, the combiner verifies each signature-share and its proof of correctness (also called the proof of robustness) using a non-interactive version of a protocol due to Chaum and Pedersen [27]) and combines only valid signature-shares into a final signature. King [28] proposes another efficient threshold signature scheme based on RSA. However, the scheme is not robust, i.e., malicious participants in this scheme can inject incorrect signature-shares to make the final signature erroneous. Fouque and Stern [29] propose a way to avoid using a safe-prime RSA modulus in the proof of robustness of [21], such that the proof remains correct even after using an RSA modulus which is not a product of two safe primes. Damgård and Koprowski [30] propose a robust threshold RSA scheme without a trusted dealer. However, the robustness of their scheme depends on a non-standard intractability assumption (in addition to the security of the standard RSA scheme). Damgård and Dupont [31] later improve the scheme [30] by removing its reliance on non-standard intractability assumptions while enjoying similar efficiency.

Langford [20] proposes a DSS-based k-out-of-l (for $k \geq t+1$) threshold signature scheme which does not rely on a single trusted dealer for computing and distributing key-shares to the participants. Instead, the scheme employs three share-generation centers in order to avoid a single-point failure. Langford also proposes another threshold signature scheme to avoid certain precomputed tables of one-time shares required in the earlier scheme, but this scheme requires $k = O(t^2)$ signers to produce a signature while tolerating only t corrupted parties. Gennaro et al. [32] provide a robust threshold signature scheme for DSS with $k = 2t+1$ signers required. They provide a security proof for their scheme based on the unforgeability of DSS signatures. Gennaro et al. [33] propose a threshold signature scheme for DSA/ECDSA using an additively homomorphic encryption scheme like Paillier encryption [34] and zero-knowledge (ZK) arguments [35]. Lindell and Nof [36] propose a fast and secure multi-party ECDSA with a practical distributed key generation technique. It makes use of a multiplication functionality and instantiates this functionality using either Paillier encryption or oblivious transfer (OT) [37, 38]. Doerner et al. [39] recently propose an efficient threshold ECDSA signature scheme whose security is derived from the computational Diffie-Hellman assumption in the global random oracle model [40].

Some other computationally expensive threshold signatures include those constructed using threshold fully homomorphic encryption [41, 42] and identity-based threshold signatures using bilinear pairings [43].

[1] A prime p is called a safe prime if it is of the form $p = 2p' + 1$, where p' is a prime as well.

2.3. Notation Used in This Article

We take λ to be the security parameter. We use $a \xleftarrow{R} S$ to denote an element a chosen from a set S uniformly at random. We denote the set of integers and the set of congruence classes of integers modulo a positive integer n (or the ring of integers modulo n) by $\mathbb{Z}$ and $\mathbb{Z}_n$, respectively. The elements of $\mathbb{Z}_n$, which are coprime to n, constitute the set $\mathbb{Z}_n^*$. The greatest common divisor of two integers a and b is denoted by $\texttt{gcd}(a, b)$. Given two integers a and b (where $a \leq b$), we denote the set $\{a, a+1, \ldots, b\}$ also by $[a, b]$. The (ordered) concatenation of two strings s_1 and s_2 is denoted by $s_1||s_2$.

2.4. Jacobi Symbol

For two integers $a \geq 0$ and $n > 0$ such that $\texttt{gcd}(a, n) \neq 1$, a is called a quadratic residue modulo n if there exists a $y \in \mathbb{Z}_n^*$ satisfying $a \equiv y^2 \mod n$ (on the other hand, a is called a quadratic non-residue modulo n if there exists no $y \in \mathbb{Z}_n^*$ satisfying $a \equiv y^2 \mod n$). Given an integer $a \geq 0$ and an odd prime p, the Legendre symbol [44] is defined as:

$$\left(\frac{a}{p}\right)_L = \begin{cases} 1, & \text{if } a \text{ is a quadratic residue modulo } p \\ -1, & \text{if } a \text{ is a quadratic non-residue modulo } p \\ 0, & \text{if } \texttt{gcd}(a, p) \neq 1. \end{cases} \tag{1}$$

Let n be an odd positive integer with the prime factorization $n = \prod_{i=1}^{m} p_i^{e_i}$, where each p_i is an odd prime. Then, for all $a \geq 0$, the Jacobi symbol [44] is defined as:

$$\left(\frac{a}{n}\right)_J = \prod_{i=1}^{m} \left(\frac{a}{p_i}\right)_L^{e_i}. \tag{2}$$

The Jacobi symbol is same as the Legendre symbol in case n is an odd prime.

3. System Model

Deploying multiple PDPs across an enterprise network eliminates the need to trust a single PDP for access validation. Trust distribution is necessary due to the following reasons. In case a there is a single PDP and it is compromised, the PDP can collude with a malicious user/device to approve its request to access a (possibly sensitive) enterprise resource that the user/device is not allowed to access otherwise. As the gateway corresponding to the resource allows all connections approved by the PDP, the possibly malicious user/device gets an access to the resource.

In order to enforce trust distribution among multiple PDPs, the agent installed at the device side communicates with the PDPs to get its connection request approved by them. Each of the PDPs then runs the trust algorithm and sends back a "partial" approval (in the form of an authenticator) if and only if the trust algorithm approves the connection request. If the agent collects authenticators from "sufficiently many" PDPs (say, at least k PDPs, for

a system parameter k), then the agent can use these authenticators in order to "convince" the resource gateway to connect that user/device with the resource. On the other hand, a malicious user/device cannot get an invalid connection accepted by the resource gateway even if it compromises any t PDPs, for any value of $t < k$.

3.1. Assumptions

We state the following assumptions that a multi-PDP zero-trust enterprise network considered in this work relies on.

- An enterprise typically has one or more trusted network administrators who are responsible for setting up the enterprise network initially and maintaining the system thereafter. Our solution also assumes such an administrator $\mathcal{B}$ for an enterprise network.
- The agent installed on a device has the complete list of all PDPs present in the enterprise network (e.g., the agent, after its deployment, can query a directory service to get the list and then continue updating the list by polling that service periodically), such that it can directly communicate with each of these PDPs.
- The total number of PDPs in the network that can be corrupted is denoted by t. The value of t must be less than that of the system parameter k, the minimum number of PDPs required to approve a connection request.
- Every connection request in ZTA must be checked for authenticity (where the user/device sending the connection request authenticates itself using its secret information, such that a PDP can check if the user/device is authentic) and authorization (where a PDP checks if the user/device is permitted to access the requested resource and authorizes the request using authenticators such as signatures). Moreover, all traffic flow must be encrypted such that only the intended receivers can decrypt the messages. In order to cope up with the requirements mentioned above, our solution relies on an enterprise public key infrastructure (PKI) for maintaining certificates issued to all entities in the enterprise network (e.g., users, devices, resource gateways) [12]. We assume that the network administrator generates and distributes these certificates to all the entities on behalf of the enterprise. These certificates are the respective public keys signed by the administrator.
- Given a connection request, a PDP runs the trust algorithm at its end in order to validate if the respective user/device is authentic and if it is authorized to access the requested resource based on certain policies (see Section 2.1). To decide the validity of the request, the trust algorithm also takes as input user-, system- and policy-databases containing up-to-date information about users/devices/policies.

For multiple PDPs employed in the network, each PDP runs an instance of the trust algorithm. The inputs fed to these instances can be either centralized (each instance of the trust algorithm uses the same databases which are centrally maintained by the enterprise) or distributed (each PDP maintains different replicas of the databases and uses them as input to the trust algorithm it runs). Each of these strategies has certain advantages (and disadvantages) over the other. Centralized databases [45] are easy to maintain (and they have a consistent view across all the PDPs), but they are vulnerable to single point of corruption/failure. On the other hand, distributed replicas of the databases are resilient against single point of corruption/failure, but achieving a synchronized and consistent view of these instances is complex in practice. We do not discuss this here in details, as our main focus in this work is to distribute the ability to authenticate a connection request among multiple PDPs. It is up to an enterprise to design, implement and maintain these databases as per its preferences. Our solution works for any of these strategies.

3.2. Threat Model

We assume that a user/device and some of the PDPs can be malicious, with a constraint that the total number of malicious PDPs is bounded by $t < k$. A malicious user/device can in turn corrupt the agent installed on that device. So, unless stated otherwise, we say that the agent itself is malicious to denote that the corresponding user/device is malicious. An agent and some of the PDPs can act maliciously as follows.

- After checking the authenticity of an agent and whether it is authorized to access the requested resource, each PDP approving the connection request authenticates its approval (by producing an authenticator), such that the agent can later produce the authenticator (or a combination of all such authenticators) to the resource gateway for validity check. A malicious agent can attempt to access an enterprise resource that it is not authorized to access otherwise. Moreover, the malicious agent can compromise or collude with some of the PDPs in order to produce a valid authenticator for an invalid connection request, such that the authenticator later passes the check at the resource gateway.

- Even if a connection request is actually valid and authorized, a malicious PDP can produce an erroneous authenticator (due to compromise/corruption of one or more of its parts, e.g., the trust algorithm), such that the check fails at the gateway and the access is denied.

There are some other attack vectors that might compromise the authenticity and confidentiality of a zero-trust enterprise network as follows.

- While sending a connection request in order to access an enterprise resource, an unauthorized agent can try to impersonate another agent (i.e., another user/device) which is authorized to access that particular resource.

- A third party can attempt to sneak in through the communication channel between two entities (e.g., a user/device and a remote enterprise server) and to learn the (possibly sensitive) information exchanged between them.

4. Our Solution

In this section, we employ the idea of trust distribution among multiple PDPs in a zero-trust enterprise network. We recall that the PDP sits in between an agent (representing a user/device) and an enterprise resource, and it decides if the connection request from the agent can be granted for that particular resource. Each of the enterprise resources is equipped with a gateway that sets up a connection with the respective resource if and only if the PDP approves that particular connection request. In Section 4.1, we outline our generic solution, that is based on techniques from k-out-of-l threshold cryptography [14], to mitigate the threats mentioned in Section 3.2. Based on this idea, we provide a concrete solution called `DistriTrust` in Section 4.3, where we use a particular threshold signature scheme for generating authenticators. In Section 4.2, we compare different threshold schemes based on the latency involved in these schemes and identify a suitable scheme for ZTA that is later used in `DistriTrust`.

4.1. Generic Solution

Instead of relying on a single PDP for authenticating connection requests, we consider l PDPs $\{P_1, P_2, \ldots, P_l\}$ that are responsible for granting (or rejecting) access to various enterprise resources. A gateway corresponding to a resource now sets up a connection if and only if at least k PDPs $\{P_{i_1}, P_{i_2}, \ldots, P_{i_k}\}$ authenticate in favor of the connection request, where $i_1, i_2, \ldots, i_k \in [1, l]$. The idea is to share (parts of) the secret key needed for authentication among the PDPs, such that any k PDPs can collaborate to generate a valid authenticator for a connection, whereas any $t < k$ number of PDPs cannot produce such a valid authenticator. In particular, each of the k PDPs uses its (secret) key-share to generate a share of the authenticator — all of these k shares are finally combined to authenticate the connection request.

4.2. Finding a Threshold Signature Scheme Suitable for Concrete Solution

In this section, we discuss some threshold signature schemes based on the latency involved and then identify a threshold signature scheme which is suitable for our concrete solution. We first discuss the latency of the following schemes with a trusted dealer who generates a secret key-public key pair for the system, computes the shares of the secret key for l participants (e.g., l PDPs in our scenario), and distributes these shares to the respective participants.

- Gennaro et al. [18] propose a solution for robust $l/2$-out-of-l threshold RSA signatures where the robustness property is achieved by employing extensions of the information checking protocol of [22, 23] and an undeniable signature [24]. This requires the dealer to generate additional data for

each of the participants in the setup phase which later help in verifying the signature-shares. [18] also presents a non-interactive signature-share verification scheme that is not based on random oracles [46] — which requires a special relationship between the sender and recipient of a signature-share. However, their threshold signature scheme uses two-level secret sharing: the i-th participant possesses a share d_i, such that all d_i's sum up to the secret RSA decryption exponent d; and each d_i is again a verifiable secret shared among the participants.

- The threshold signature scheme proposed by Shoup [21] is as efficient as possible for a robust threshold scheme based on RSA signatures [30] — it uses only one level of secret sharing and each participant simply sends a single signature-share to the combiner in order to contribute to the signing of a connection request. No further interaction is needed among the participants. The only requirements are that $k \geq t+1$ and $l - t \geq k$, where t is the maximum number of corrupted participants allowed. During the setup phase, the dealer needs to compute l evaluations of a $(k-1)$-degree polynomial and l exponentiations. Given a secret key-share by the dealer, each participant needs to compute two hashes and four exponentiations to compute a signature-share. The signature-shares are validated by a combiner. Upon receiving k signature-shares, the combiner requires total $O(k)$ exponentiations to verify the signature-shares and to combine them into a final signature. We note that some other threshold signature schemes based on RSA [15, 28] with a trusted dealer provide slightly more efficient solution than [21], but they are not robust, i.e., malicious participants can inject incorrect signature-shares to make the final signature erroneous. We also note that the dealer is required to interact with the participants only once (i.e., in the setup phase).

- The threshold RSA signature scheme proposed by Rabin [19] is similar to [21], but the major drawback of the scheme is that for each signature generation, all l signature-shares are required. If some shares do not pass verification, then these shares must be reconstructed using a verifiable secret sharing scheme.

- In the DSS-based threshold signature scheme proposed by Langford [20], the dealer needs to compute one exponentiation modulo a prime p_1. During the generation of signature-shares, the dealer does $O(l)$ evaluations of two $(k-1)$-degree polynomials to compute shares of k' and $k'x'$, where k' is a nonce and x' is the secret signing key. Given two shares by the dealer, each participant needs to compute one hash, and two multiplications and one addition modulo another prime p_2 in order to generate its signature-share. Combining the signature-shares requires $O(k)$ multiplications and $O(k)$ additions modulo p_2. The scheme has two drawbacks: it is not robust (i.e., the final signature is invalid if the partial signatures contributed by some of the participants are incorrect); the dealer has to compute and distribute shares of k' and $k'x'$ for each signature, since no

two signatures can use the same value of k'. Thus, the unused values of k' must be tracked — which is cumbersome: the dealer may use precomputed tables and each participant must check if the same k' has been used before. [20] proposes another scheme without any trusted dealer and precomputed tables, but it requires signature-shares from $k = O(t^2)$ participants instead of $k = O(t)$ of them, where t is the maximum number of corrupted participants allowed.

There are threshold signature schemes without a trusted dealer where the participants collaboratively compute their shares of a secret key, generate the corresponding public key for the system, and compute the final signature by interacting with each other. In general, as these schemes require many rounds of communication among the participants, the latency involved is much higher compared to the schemes relying on a trusted dealer.

- *Variants of RSA-based threshold signature schemes (without a trusted dealer)*: The efficient RSA-based scheme with a trusted dealer [21] considers an RSA modulus which is the product of two safe primes. For distributed RSA key generation without a trusted dealer, it is hard to ensure that the modulus has this property. Miyazaki et al. [26] build an RSA-based threshold signature scheme using a key generation protocol that does not produce such a modulus. It uses two-level secret sharing and needs interaction between participants for generating each signature — which makes the scheme significantly less efficient than [21]. Fouque and Stern [29] propose a way to avoid a safe-prime RSA modulus in the proof of correctness of [21]. However, they show that their enhancements decrease the performance of the basic protocols of [21]. Damgård and Koprowski [30] propose a robust threshold RSA scheme without a dealer which is as efficient as [21] except the additional rounds of communication needed to generate key-shares in a distributed way. Moreover, the robustness of the scheme depends on a non-standard intractability assumption. Damgård and Dupont [31] improve the scheme [30] by removing its reliance on any non-standard intractability assumptions while enjoying similar efficiency. However, the extra rounds of communication are still required in the setup phase.

- *Variants of threshold signature schemes based on discrete logarithm assumptions (without a trusted dealer)*: There are many threshold signature schemes, based on discrete logarithm assumptions, which operate without a trusted dealer. All of these schemes require several rounds of communication both in the setup phase and the signing phase.

  — The setup phase in [32] requires a joint secret sharing between participants, where each participant has to compute $O(l)$ evaluations of a $(k-1)$-degree polynomial and send respective shares to each of the other participants using a point-to-point communication. In addition, each participant has to perform modular exponentiations and interpolation in the

exponent. In the signing phase, the participants need to jointly execute protocols like secret sharing protocol, reciprocal protocol and multiplication protocol where each participant has to perform $O(l + k)$ modular exponentiations and communicate with other participants using either broadcasting or point-to-point communication. Moreover, given a value of t such that t is the maximum number of corrupted participants allowed, it is necessary to share the secret key among at least $(2t + 1)$ participants and at least $(2t + 1)$ participants are required to produce a signature.

— The setup phase in [33] involves a joint threshold-key generation for an additively homomorphic encryption scheme, and other per-participant operations like commitment generation and its verification, exponentiations, and computing and verifying zero-knowledge (ZK) arguments. This involves several rounds of broadcasting or point-to-point communication. During the signature-generation phase, the participants need to perform encryptions, commitment generation and verification, exponentiations, and computing and verifying ZK arguments, and joint decryption; and these operations also require several rounds of communications among the participants.

— The fast and secure multi-party ECDSA proposed by Lindell and Nof [36] makes use of a multiplication functionality and instantiates this functionality using either Paillier encryption or oblivious transfer (OT). The OT-based protocol requires: $(11 + 331l)$ elliptic-curve (EC) multiplications during the the setup, $(83 + 78l)$ EC multiplications during each signing, five rounds of communications during the setup, eight rounds of communications during each signing. The Paillier-based protocol requires: $(11 + 11l)$ elliptic-curve (EC) multiplications during setup, $(83 + 78l)$ EC multiplications during each signing, $(11 + 11l)$ large-integer exponentiations during the setup, $21l$ large-integer exponentiations during each signing, five rounds of communications during the setup, eight rounds of communications during each signing.

— The threshold ECDSA signature scheme proposed by Doerner et al. [39] involves sub-protocols such as 2-party-multiplication, k-party modular inverse sampling, ECDSA-Setup and ECDSA-Sign. The costs involved in the setup phase are as follows. For each pair of participants P_i and P_j: 2-party-multiplication takes five rounds of communications, P_i does five EC multiplications and six hashes, P_j does four EC multiplications and seven hashes. A k-party modular inverse sampling takes five rounds of communications; each party does $O(l)$ EC multiplications and $O(l)$ hashes. ECDSA-Setup takes five rounds of communications; each party does $O(k^2 + lk + l)$ EC multiplications and $O(l)$ hashes. The costs involved in each signing phase are as follows. For each pair of participants P_i and P_j: 2-party-multiplication takes three rounds of communications, each of P_i and P_j does $O(m')$ hashes, where the group elements are represented in m' bits. A k-party modular inverse sampling takes $(\log(k) + 6)$ rounds of communications; each party does three EC multiplications and $O(m'k)$ hashes.

ECDSA-Sign takes $(\log(k)+6)$ rounds of communications; each party does six EC multiplications and $O(m'k)$ hashes.

Suitable threshold signature for our solution. Based on the latency involved in different threshold signature schemes discussed above, we choose the scheme proposed by Shoup [21] as a suitable candidate for our concrete solution due to the following reasons.

1. The threshold signature scheme in [21] is robust against malicious participants (PDPs in our case) which may attempt to make the final signature invalid by providing incorrect signature-shares. The scheme is more efficient compared to other robust threshold signature schemes discussed above.

2. The PDPs need not communicate with each other during the setup phase in order to compute their respective key-shares. The significant overhead due to such additional rounds of communication required in many threshold signature schemes is not desired in ZTA, where connection requests must be handled in real-time. We note that [21] relies on a trusted dealer which initially computes and distributes the key-shares to respective participants. As ZTA provides security solutions for an enterprise and an enterprise typically has a trusted network administrator, we assume that the administrator works as the dealer. We stress that the distribution of key-shares among the PDPs is a *one-time job* which is independent of the future connection requests.

3. As the procedure run by a PDP to generate its signature-share is independent of the other PDPs, the PDPs need not communicate with each other while computing the respective signature-shares. Thus, given a connection request, the signature-shares can be produced by the PDPs in parallel — which roughly takes the time required by just one PDP to produce its respective signature-share.

4. The proof of correctness computed by a PDP for its signature-share is generated non-interactively; that is, it is derived without any challenge-response protocol between the agent and the PDP. In order to generate the proof without interacting with the agent, the PDP uses a hash function modeled as a random oracle [46] — which saves a round of communication between them.

4.3. `DistriTrust`: *A Concrete Solution*

In this section, we provide a concrete solution that we call `DistriTrust`. `DistriTrust` is based on a threshold RSA signature scheme [21] which in turn uses the traditional RSA signature scheme [16] and polynomial-based secret sharing techniques [17]. As discussed in Section 3.1, we rely on an enterprise PKI for maintaining certificates for different entities of the enterprise. These certificates can be respective public keys signed by the network administrator

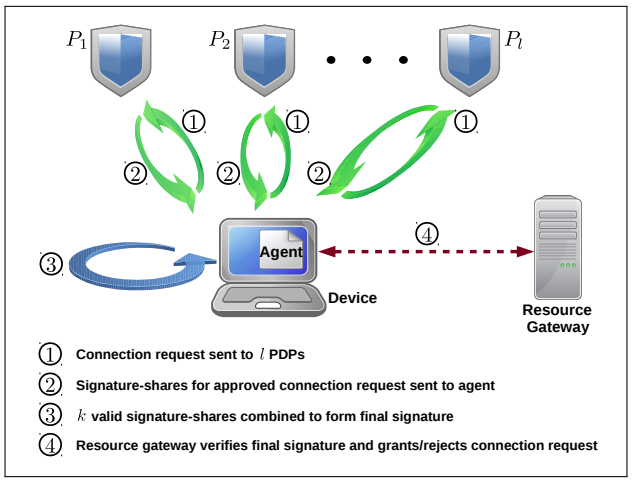

Figure 2: Access validation of a connection request in `DistriTrust`.

$\mathcal{B}$ of the enterprise. Thus, every entity in `DistriTrust`, including the administrator $\mathcal{B}$, is identified by its public key (say, pk), and the entity holds the corresponding secret key (say, sk). These key pairs are used for providing confidentiality and authenticity of the connection requests in the form of public-key encryptions/signatures. Moreover, for a session between any two entities in the network, the key pairs can also be used to compute a symmetric session key (e.g., using Diffie-Hellman key exchange [47]) shared between these entities. We assume that every session in `DistriTrust` is encrypted and authenticated using a session key with the help of an authenticated encryption scheme $\mathcal{E}$ (similar to the standard Transport Layer Security protocol [48]; e.g., TLS 1.3 uses AEAD_AES_128_GCM [49] as one of the authenticated encryption schemes).

The network administrator $\mathcal{B}$ initially computes and distributes shares of a secret key (that are later used for generating a threshold signature) to l PDPs $\{P_1, P_2, \ldots, P_l\}$. For a connection request sent by an agent $\mathcal{A}$, valid signature-shares computed by any group of k PDPs $\{P_{i_1}, P_{i_2}, \ldots, P_{i_k}\}$ can be combined by $\mathcal{A}$ in order to produce a signature on the approved connection request (see Figure 2). Finally, the agent $\mathcal{A}$ sends the final signature to the gateway corresponding to the requested resource for verification, and the gateway grants/denies the request depending upon the validity of the signature. We describe different phases of `DistriTrust` in details as follows.

Setup phase. In the setup phase, the network administrator $\mathcal{B}$ computes and distributes the shares of a secret key among $\{P_1, P_2, \ldots, P_l\}$ as follows.

- $\mathcal{B}$, having a public key-secret key pair $(pk_{\mathcal{B}}, sk_{\mathcal{B}})$, initially chooses two safe primes p and q of equal length (the length is decided based on the security parameter λ) uniformly at random, where $p = 2p' + 1, q = 2q' + 1$ for two other primes p', q'. $\mathcal{B}$ takes $n = pq$ as the RSA modulus and chooses another prime $e > l$ as the RSA public exponent. $\mathcal{B}$ signs $\mathtt{PK} = (n, e)$ using

$sk_{\mathcal{B}}$ and sets it as the public key for the threshold signatures, such that every entity in the system can check the authenticity of the (threshold) public key $\texttt{PK}$ using $pk_{\mathcal{B}}$.

- $\mathcal{B}$ computes the decryption exponent $d \in \mathbb{Z}_m$, where $de \equiv 1 \mod m$ for $m = p'q'$. $\mathcal{B}$ constructs a $(k-1)$-degree polynomial $f(X) = \sum_{i=0}^{k-1} a_i X^i$, where $a_0 = d$ and $a_i \xleftarrow{R} \mathbb{Z}_m$ for each $i \in [1, k-1]$. $\mathcal{B}$ sets $\texttt{SK} = (p, q, d)$ as the secret key for the threshold signatures.

- For each $i \in [1, l]$, $\mathcal{B}$ computes

$$s_i = f(i) \cdot \Delta^{-1} \mod m \tag{3}$$

as the secret-key share for the i-th PDP P_i, where $\Delta = l!$ (i.e., Δ is the factorial of l). $\mathcal{B}$ then distributes the (secret) key-shares to the respective PDPs.

- Let Q_n be the subgroup containing the quadratic residues present in $\mathbb{Z}_n^*$ (a quadratic residue can be written as a square of another element). $\mathcal{B}$ chooses a random element $v \xleftarrow{R} Q_n$ and another element $u \in \mathbb{Z}_n^*$ such that the Jacobi symbol $(\frac{u}{n})_J = -1$ (see Section 2.4). For each $i \in [1, l]$, $\mathcal{B}$ computes $v_i = v^{s_i} \in Q_n$. Then, $\mathcal{B}$ sets $\texttt{VK} = (v, u)$ as the verification key and $\texttt{VK}_i = v_i$ as the verification-key share corresponding to P_i (for each $i \in [1, l]$), and makes these keys public after signing them using $pk_{\mathcal{B}}$.

Phase for generating signature-shares. In this phase, the agent $\mathcal{A}$ representing a user/device sends a connection request to the PDPs, and some of the PDPs use their respective key-shares to generate signature-shares approving the connection request.

$\mathcal{A}$ embeds the connection information `conn_info` in a connection request and sends it to the i-th PDP P_i for each $i \in [1, l]$. Suppose l_1 PDPs (for $k \leq l_1 \leq l$) are online at that time and find the connection request valid after running the trust algorithm at their end. Without loss of generality, we assume that these l_1 PDPs are identified by the first l_1 indices, i.e., $[1, l_1]$. For each $i \in [1, l_1]$, P_i computes a signature-share on the message $M = \texttt{conn\_info}$ as follows and sends its signature-share to $\mathcal{A}$ which later combines k valid shares to obtain the final threshold signature.

- P_i computes $\hat{x} = H(M)$, where $H : \{0,1\}^* \to \mathbb{Z}_n^*$ is a hash function mapping messages of arbitrary lengths to the elements of $\mathbb{Z}_n^*$, and defines x as follows:

$$x = \begin{cases} \hat{x}, & \text{if } (\frac{\hat{x}}{n})_J = 1 \\ \hat{x}u^e, & \text{if } (\frac{\hat{x}}{n})_J = -1. \end{cases} \tag{4}$$

- The signature-share generated by P_i consists of: 1) an element

$$x_i = x^{2s_i} \in Q_n, \tag{5}$$

and 2) a proof of correctness Π_i (that is generated using the non-interactive variant of a protocol due to Chaum and Pedersen [27]) as defined below. Let $L(n)$ be the size of n when represented in bits. Let $H' : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a hash function mapping messages of arbitrary length to λ-bit integers for the security parameter λ. In order to construct the proof of correctness for its signature-share, P_i chooses a random number $r \xleftarrow{R} \{0, 1, \ldots, 2^{L(n)+2\lambda} - 1\}$ and computes $v' = v^r, x' = \tilde{x}^r, c = H'(v, \tilde{x}, v_i, x_i^2, v', x'), z = s_i c + r$, where $\tilde{x} = x^4$. The proof of correctness associated with the signature-share is $\Pi_i = (z, c)$.

We note that Π_i is later used by the agent $\mathcal{A}$ to verify whether P_i has used the same key-share s_i (which was used to derive the verification-key share v_i from the verification key v in the setup phase) to compute its signature-share x_i. Given the pair (x_i, Π_i), the verification can be done by checking whether

$$c \stackrel{?}{=} H'(v, \tilde{x}, v_i, x_i^2, v^z v_i^{-c}, \tilde{x}^z x_i^{-2c}) \tag{6}$$

with the help of the verification keys v and v_i.

Phase for combining signature-shares. $\mathcal{A}$ takes the signature-shares on the message $M =$ `conn_info` sent by any k PDPs, such that their respective proofs of correctness satisfy Eqn. 6 individually. $\mathcal{A}$ then combines these shares to generate the final signature $\hat{y}$ as follows.

- $\mathcal{A}$ chooses a set of k PDPs $\{P_{i_1}, P_{i_2}, \ldots, P_{i_k}\}$ identified by a set of indices $S = \{i_1, i_2, \ldots, i_k\} \subseteq [1, l_1] \subseteq [1, l]$. Then, $\mathcal{A}$ takes x_{i_j} for all $j \in [1, k]$ and computes

$$w = x_{i_1}^{2\psi_{0,i_1}^S} \cdot x_{i_2}^{2\psi_{0,i_2}^S} \cdots x_{i_k}^{2\psi_{0,i_k}^S}, \tag{7}$$

where $\psi_{i,j}^S = \Delta \cdot \frac{\prod_{j' \in S \setminus \{j\}} (i - j')}{\prod_{j' \in S \setminus \{j\}} (j - j')} \in \mathbb{Z}$ for $i \in [0, l] \backslash S$, $j \in S$ and $\Delta = l!$.

- $\mathcal{A}$ takes $e' = 4$. We note that e is a prime greater than l (where $l \geq 2$); thus it follows that $\texttt{gcd}(e, e') = 1$. $\mathcal{A}$ runs the extended Euclidean algorithm[2] on (e', e) and computes a and b such that $e'a + eb = 1$. $\mathcal{A}$ computes $y = w^a x^b$ and produces the signature $\hat{y}$ such that

$$\hat{y} = \begin{cases} y, & \text{if } (\frac{\hat{x}}{n})_J = 1 \\ yu^{-1}, & \text{if } (\frac{\hat{x}}{n})_J = -1. \end{cases} \tag{8}$$

[2] Given two positive integers n_1 and n_2 as input, the classical Euclidean algorithm computes $\texttt{gcd}(n_1, n_2)$. Given the same integers as input, the *extended* Euclidean algorithm [50] also computes two integers c_1 and c_2, such that $n_1 c_1 + n_2 c_2 = \texttt{gcd}(n_1, n_2)$.

Finally, $\mathcal{A}$ sends the signature $\hat{y}$ along with M to the gateway.

Phase for signature verification. After receiving M and $\hat{y}$ from $\mathcal{A}$, the gateway computes $\hat{x} = H(M)$ and checks if

$$\hat{y}^e \stackrel{?}{=} \hat{x}. \tag{9}$$

If the equality in Eqn. 9 holds, the gateway sets up a connection between the requesting device and the resource; otherwise, it rejects the connection request.

5. Correctness and Security

In this section, we discuss the correctness and security of `DistriTrust` described in Section 4.3.

Correctness. From the Lagrange interpolation formula [51], we have

$$\begin{aligned} f(i) &= \sum_{j \in S} f(j) \cdot \frac{\prod_{j' \in S \setminus \{j\}} (i - j')}{\prod_{j' \in S \setminus \{j\}} (j - j')} \mod m \\ &= \sum_{j \in S} f(j) \cdot \Delta^{-1} \cdot \psi_{i,j}^S \mod m \\ &= \sum_{j \in S} s_j \cdot \psi_{i,j}^S \mod m, \qquad \text{[from Eqn. 3]} \end{aligned} \tag{10}$$

where $\psi_{i,j}^S = \Delta \cdot \frac{\prod_{j' \in S \setminus \{j\}} (i - j')}{\prod_{j' \in S \setminus \{j\}} (j - j')} \in \mathbb{Z}$ for $i \in [0, l] \setminus S$, $j \in S$ and $\Delta = l!$.

From Eqn. 7, we get

$$\begin{aligned} w &= x_{i_1}^{2\psi_{0,i_1}^S} \cdot x_{i_2}^{2\psi_{0,i_2}^S} \cdots x_{i_k}^{2\psi_{0,i_k}^S} \\ &= \prod_{j \in S} x_j^{2\psi_{0,j}^S} \\ &= \prod_{j \in S} x^{4 s_j \psi_{0,j}^S} \qquad \text{[from Eqn. 5]} \\ &= x^{4 \sum_{j \in S} s_j \psi_{0,j}^S} \\ &= x^{4f(0)} \qquad \text{[from Eqn. 10]} \\ &= x^{4d}. \qquad \text{[since } f(0) = a_0 = d\text{]} \end{aligned}$$

As $de \equiv 1 \mod m$, we have $w^e = x^{4de} = x^4 = x^{e'}$. Again, $y = w^a x^b$. Thus, $y^e = w^{ea} x^{eb} = x^{e'a} x^{eb} = x^{e'a + eb} = x$ (as $e'a + eb = 1$) — which makes y a valid RSA signature on x. This implies that $\hat{y}$ is a valid RSA signature on $\hat{x} = H(M)$.

Security. `DistriTrust` satisfies the following security properties which are essential for a zero-trust network.

- **Authenticity.** In `DistriTrust`, every connection request is checked for authenticity — the trust algorithm executed by each PDP checks the validity of the user/device. Based on the access policy of an enterprise resource, single- or multi-factor authentication of users is done using their identities such as passwords and biometrics (e.g., fingerprints, facial recognition). Device authentication is done using the device identity such as device certificates. Thus, an unauthorized user/device cannot impersonate another user/device which is authorized to access a particular enterprise resource.

  Each PDP, that approves a connection requested by an agent, sends the agent a signature-share such that the agent can later combine the shares into a signature and send it to the gateway. A PDP can be malicious in the sense that it may produce an erroneous signature-share such that, if included in the final signature, the signature-share makes the final signature invalid. Consequently, the signature does not pass verification at the gateway, even if the connection request is valid and more than k honest PDPs are available (a denial-of-service/DoS attack). In order to prevent this, `DistriTrust` employs a robust threshold signature scheme [21], where each PDP P_i is required to produce a proof of correctness Π_i (along with its signature-share x_i) that the agent later verifies before including x_i in the final signature. The proof of correctness Π_i validates if P_i has correctly computed its signature-share x_i using the same (secret) key-share s_i provided by the network administrator $\mathcal{B}$ in the setup phase.

- **Authorization.** In `DistriTrust`, the PDPs validate if a user/device is authorized to access an enterprise resource. In case a connection request from an agent representing a user/device is valid and authorized, a PDP produces a signature-share for the connection request. Due to the unforgeability property of the underlying threshold signature scheme [21], a malicious agent cannot obtain a valid final signature for an unauthorized access, even if it colludes with any $t < k$ dishonest PDPs. In other words, if the corresponding gateway receives a connection request along with a valid signature authorizing the request, it is implied that at least k PDPs have approved the request and the user/device is thus authorized to access the enterprise resource.

- **Confidentiality.** The confidentiality of the (possibly sensitive) messages exchanged between a pair of entities in a session is guaranteed by the authenticated encryption scheme $\mathcal{E}$ employed in `DistriTrust`, which ensures that only the intended recipient can decrypt their corresponding messages. Thus, a third party, without having the corresponding secret key, cannot sneak in through a communication channel between a pair of entities and learn the information being exchanged.

6. Performance Analysis

6.1. Asymptotic Analysis

Computational cost. The network administrator $\mathcal{B}$ evaluates a $(k-1)$-degree polynomial l times (for computing l shares of the secret key) and performs l exponentiations (for computing the respective shares of the verification-key). We note that this cost is incurred only once during the setup phase. In the signing phase, each PDP needs to perform two hashes and four exponentiations in order to compute a signature-share. Given k signature-shares, the agent performs $O(k)$ exponentiations to verify these shares and to combine them into a final signature. Finally, the gateway performs one hash and one exponentiation to verify the final signature.

Unlike the generation of signature-shares that can be performed by the PDPs in parallel, the computational cost incurred at the agent side for checking proofs of correctness for the individual signature-shares and combining these shares is proportional to the value of the parameter k. However, we note that verifying $O(k)$ proofs of correctness and performing $O(k)$ exponentiations required to compute w (see Eqn. 7) can be potentially parallelized depending on the hardware capability of the respective device. On the other hand, the (constant) cost for cryptographic operations at each PDP (required for generating the signature-share) and the gateway (required for verifying the final signature) is much less compared to that required by the agent.

We note that generation (and verification) of a proof of correctness consumes a major share of the total computational cost required for generation (and verification) of a signature-share (see Figure 3). However, as we have discussed above, proofs of correctness are required to make the access validation robust against an attacker trying to prevent honest PDPs to produce a valid signature for a valid connection request.

Communication cost. Let $|n|$ and $|m|$ be the size of n and the size of m, respectively, which depend on the value the of security parameter λ. For example, let us consider the value of λ to be 80 (i.e., the system provides 80-bit security). In that case, both $|n|$ and $|m|$ are around 128 bytes (to be precise, 1024 bits and 1022 bits, respectively). Let `conn_info` be encoded using β bytes. We estimate the communication bandwidth required in `DistriTrust` based on these parameters as follows.

In the setup phase, the network administrator $\mathcal{B}$ sends a (secret) key-share s_i (an element modulo m) to each P_i — which requires around 128 bytes of bandwidth for each $i \in [1, l]$. During the signing phase, an agent sends `conn_info` to each P_i — which requires around β bytes of bandwidth for each $i \in [1, l]$. Moreover, each P_i sends a signature-share x_i (an element modulo n) and a proof of correctness $\Pi_i = (z, c)$ (where the size of z is roughly same as $|m|$ and c is a λ-bit string) to the agent — which accounts for around 356 bytes of bandwidth for each $i \in [1, l_1]$. Finally, the agent sends the final signature (an element modulo n) and `conn_info` to the gateway — which requires around $(128 + \beta)$ bytes of bandwidth.

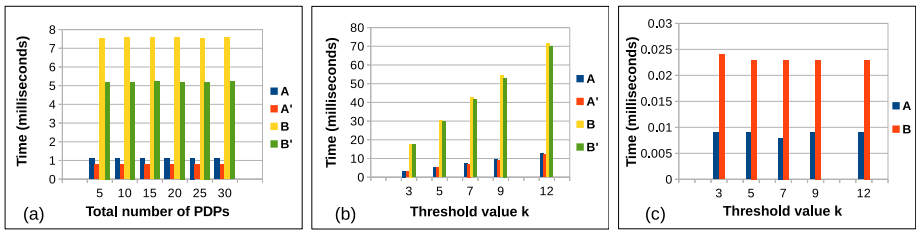

Figure 3: Time required by different entities involved in access validation in `DistriTrust` for: (**A**) $\lambda = 80$ and (**B**) $\lambda = 112$. (a) Total time required by a PDP to generate a signature-share and its proof of correctness. (b) Total time required by the agent to validate k proofs of correctness and combine k signature-shares into a signature. (c) Time required by the gateway to verify a signature.
A': Time required by a PDP in (a) (or by the agent in (b)) for generation (or verification) of proof(s) of correctness for $\lambda = 80$ — which is included in the total time required for **A**.
B': Time required by a PDP in (a) (or by the agent in (b)) for generation (or verification) of proof(s) of correctness for $\lambda = 112$ — which is included in the total time required for **B**.

In order to reduce the network congestion (which in turn reduces the network latency), we can employ another system parameter $1 \leq \alpha \leq \frac{l}{k}$ such that: if the agent communicates with around $k\alpha$ PDPs (instead of all l PDPs), it can expect that k of them will be honest. Given that the number of malicious PDPs allowed in the system is $t < k$ (where $l \geq k + t$) and assuming that these malicious PDPs are uniformly distributed among all l PDPs in the network, the value of the parameter α is $\frac{l}{l-t}$. In other words, if the agent sends a valid connection request to around $\frac{kl}{l-t}$ PDPs, then it can expect to receive the approval from around k of them. Clearly, the value of α decreases as the number of honest PDPs $(l - t)$ present in the network increases.

6.2. Experimental Results

We evaluate the performance of access validation in `DistriTrust` in terms of the computational cost incurred by the corresponding entities involved: a PDP, an agent and a gateway. We build a prototype of `DistriTrust` — the codes are written in C and are compiled using GCC (version 7.5.0). To handle large integers and various operations on them, we use the GMP library (version 6.1.2) [52]. We use the OpenSSL library (version 1.1.1) [53] for implementing the cryptographic operations needed in `DistriTrust`. We perform the experiments on a 3.60 GHz Intel i7 processor with 12 GB RAM.

In our experiments, we consider two values of the security parameter λ: 80 and 112. Depending on these values of λ, we take the size of the RSA modulus n in `DistriTrust` to be 1024 bits and 2048 bits, respectively. Each n we consider in our experiments is a product of two safe primes p and q. The hash functions H and H' are instantiated using SHA-256.

Figure 3 shows the time required by different entities in `DistriTrust`. Figure 3(a) shows the time required by a PDP to generate a signature-share and the corresponding proof of correctness. We note that the computational cost for each PDP is independent of the value of l, the number of total PDPs present in

the network. For example, given different values of l as shown in the figure, the time required is similar (in the range 1.11–1.13 ms for $\lambda = 80$, and in the range 7.55–7.58 ms for $\lambda = 112$). Similarly, the computational cost for the gateway is independent of the value of k, the threshold number of PDPs required to produce a signature. Figure 3(c) shows the time required by the gateway to verify a signature. For example, given different values of k as shown in the figure, the time required for signature verification is similar (in the range 0.008–0.009 ms for $\lambda = 80$, and in the range 0.023–0.024 ms for $\lambda = 112$). Figure 3(b) shows the time required by the agent to check the validity of k signature-shares and combine these shares into a single signature. We take the value of l to be 30 and report the time for varying values of k as shown in the figure. For example, if `DistriTrust` is required to provide 80-bit security and the threshold value is $k = 5$, then the agent requires around 5.29 ms for validating k signature-shares and combining them into the final signature.

From Figure 3(a–b), we observe that the major part of the computational cost required to generate a signature-share (at the PDP side) and the final signature (at the agent side) is due to generation and verification of proof(s) of correctness (e.g., for a PDP, around 70% of the total cost is consumed to generate a proof of correctness; whereas around 95% of the cost at the agent side is due to the verification of the proofs of correctness sent by the PDPs). We note that, instead of using proofs of correctness, the agent can itself verify the final signature before sending it to the gateway, which can be done in a very efficient way (see Figure 3(c)). However, the agent now cannot identify a malicious PDP in case the signature verification fails; so it may choose an incorrect signature-share from the same (or another) PDP again while selecting another k-subset of signature-shares. In `DistriTrust`, proofs of correctness ensure robustness of the system against malicious PDPs and allow the agent to choose correct signature-shares in the first place, such that their combination produces a valid signature. We note that, if the PDPs in a system are known to be semi-honest (i.e., they follow the protocol honestly but try to learn some secret information, e.g., the decryption exponent d) instead of being malicious, then proofs of correctness are not required. In that case, the computational cost required to generate a signature-share as well as the final signature reduces significantly (e.g., for $\lambda = 80$, a PDP would then require around 0.36 ms to generate a signature-share and the agent, on the other hand, would require around 0.23 ms to combine $k = 5$ signature-shares into the final signature).

7. Discussion

7.1. Architectural and Operational Elements Supporting `DistriTrust`

In this work, we focus on achieving resilience, in a zero-trust enterprise network, against compromise of the secret signing-key and denial-of-service/DoS attacks, issues specifically due to the PDP as a single point of failure/corruption. We present `DistriTrust` which provides the concepts and methods that an enterprise can adopt to achieve this resilience by using multiple PDPs in a threshold manner. We exclude, from this work, the actual operational details such

as the exact design, code, testing, deployment and management of the PDPs, the agents, the resource gateways and other operational elements involved in a zero-trust network, although they are of utmost importance.

A thorough implementation of a zero-trust enterprise network must have several key components such as an identity management (IDM) system, data access policies, an enterprise public-key infrastructure (PKI), a security information and event management (SIEM) system, a continuous diagnostics and mitigation (CDM) system, activity and event logs, and so on [12]. These components are crucial for a zero-trust network to work properly. For example, the IDM system manages the enterprise users' attributes, roles, groups and other information which are used in the policy decision process and directly affect the policy decisions made. Data access policies (sets of rules regarding access to enterprise resources) influence the outcome of the policy decision in order to determine whether a user gets access to perform the requested operation on an enterprise resource. Apart from these elements, there are back-end database servers which are used to store the user/device database, resource database, resource policies, threat-intelligence information and other relevant data. Many of the components mentioned above use these databases to perform their respective jobs. We note that maintaining any of these components insecurely can lead to malicious actors influencing policy decisions which may result in elevated privileges for specific users.

We recognize that securely designing, deploying and maintaining the supporting infrastructure will be a challenge for `DistriTrust`. However, we can leverage off-the-shelf and well-known commercial services for nearly all the elements of the supporting infrastructure. Many enterprises already use them to operate, secure and provide highly available services for these elements. For example, an enterprise typically deploys an IDM system on-premises (e.g., on-premises LDAP [54]), or uses a cloud-based IDM system (e.g., Microsoft Azure Active Directory or Azure AD [55]), or deploys a hybrid solution with both on-premises and cloud components. Organizations such as DigiCert [56], GlobalSign [57] and Entrust [58] provide global-scale PKI services. On the other hand, large organizations such as Apple [59] and Google [60] run their own PKI ecosystems. Access policies can be defined in expressive ways using Open Policy Agent (OPA) [61] or Casbin [62].

As mentioned earlier, many of the key components use back-end database servers that store various information regarding the enterprise users, devices, resources, policies (in the form of authorization tables and access control lists [63]), and so on. The database servers in many enterprises are typically run in a high-availability mode. For example, Microsoft's SQL Server is usually deployed after considering all the operational aspects such as high availability, "always-on" groups, failover, disaster recovery, and so on [64] (a similar procedure is followed by other enterprise database management systems such as PostgreSQL [65] and Oracle [66]). We note that `DistriTrust` requires each (honest) PDP to reach the same conclusion about an access request (i.e., whether it can be authorized or not) individually. For the same reason, the instances of these databases servers used by the PDPs must be synchronized with each other. The database

servers can be managed centrally, or replicas of these servers can be distributed among multiple PDPs, where each PDP would have its own local copies of these databases. For the latter scenario, different database/directory service providers implement distributed replicas (with synchronization) in different ways. For example, OpenLDAP (an open source implementation of LDAP) [67] creates and updates a replicated directory using LDAP sync replication (`syncrepl`) engine [68]. Database replication in MySQL 8.0 can be done in an asynchronous or semi-synchronous way [69]. IBM's iBase database replication [70] distributes and synchronizes copies of data and database objects across different instances of the SQL Server using the standard tools available in the SQL Server.

In this work, we have abstracted the description of the concepts and methods involved in `DistriTrust` from the low-level design and implementation details of the overall architectural and operational elements, since our focus is to distribute the ability to authenticate/authorize a connection request among multiple PDPs in order to prevent the single point of failure/corruption. `DistriTrust` can exploit standard off-the-shelf services, as discussed above, that are already availed by many of the existing enterprises.

7.2. Risk Metrics Associated with Number of PDPs in `DistriTrust`

We build `DistriTrust` based on a k-out-of-l threshold signature scheme, where valid signature-shares from at least k PDPs (out of total l PDPs) are needed to produce a final signature for a given connection request. This implies that k is the minimum (threshold) number of PDPs which must not be compromised. The total number l of PDPs present in a particular enterprise network is influenced by risk metrics such as the overall fault-tolerance level the enterprise wants to achieve for validating access to its resources (along with other design constraints such as deployment/management cost [71]). For example, large enterprises may employ `DistriTrust` with a higher number of PDPs in their respective networks in order to ensure better availability of its access-validation service in the presence of a higher number of failed/compromised PDPs (and also possibly to improve other functionalities, e.g., better load balancing among the available PDPs). The value of the threshold parameter k can also be attributed to various risk metrics corresponding to a given network. For example, if an enterprise can estimate, based on certain analyses/predictions, the maximum number t of PDPs that can be offline/compromised, then this value of t serves as one of the potential risk metrics. In that case, the enterprise must ensure that the value set for k is greater than the value of t; otherwise, t compromised PDPs can collaboratively produce a valid signature for any invalid connection. We note that, the higher the value of k is for a given value of l, the higher number (t) of offline/compromised PDPs the network can withstand — which increases the resilience of the network in the presence of an attacker (e.g., for $k = l$, the attacker has to compromise all the PDPs to authorize an invalid connection; whereas for $k = 1$, the attacker has to compromise just a single PDP to do the same). However, increasing the value of k may reduce the availability of the access-validation service of the network (e.g., for $k = l$, access validation cannot be done in case even a single PDP is offline; whereas for

$k = 1$, access validation can be done successfully even if a single PDP is available). Another security feature of a given network, dependent on the value of k, is the frequency of secure key-distributions among the PDPs. If the number of PDPs actually compromised in an enterprise network reaches k, then the access validation in `DistriTrust` becomes insecure as the attacker controlling these k PDPs can generate valid signatures on invalid connection requests. In that case, the corrupted PDPs must be sanitized (or excluded from the set of valid PDPs), new PDPs may be installed and included in the set of valid PDPs (if required), the respective shares of fresh secret signing keys may be distributed among all valid PDPs available at that time (if required).

We note that further development of various risk metrics, associated with the distributed and low-latency access validation in a given zero-trust enterprise network, is needed in order to appropriately size the number of PDPs while, at the same time, meeting the security requirements for that network. We leave the comprehensive study of these risk metrics as a future work.

8. Another Use Case: Integrating Threshold Techniques of `DistriTrust` into OAuth 2.0

Although `DistriTrust` is designed for distributed and low-latency access validation in a zero-trust enterprise network, its threshold techniques can also be applied/integrated in other use cases such as OAuth.

OAuth 2.0 is a standard authorization framework [72] which involves a resource owner or user (and its agent, e.g., a browser), an authorization server, a resource server (and its gateway) and a client (a third-party application, e.g., web application, native application) seeking access to the resource server on behalf of the user. The OAuth 2.0 protocol has a prevalent usage for delegating authorization to a client. The flow of the protocol is as follows. The client gets an *access token* from the authorization server after the user authorizes the client to access the requested resource (possibly) in a restricted manner. Then, the client sends the access token to the gateway of the resource server for verification before the client is granted access to the requested resource. Access tokens eliminate the requirement of sharing secret credentials (e.g., passwords) of the user with the client. An access token is often *self-contained* in the sense that the content of the token is signed by the authorization server, such that the gateway can verify the signature using only the public key of the authorization server and it does not have to consult any back-end database to validate the access token. JSON Web Tokens (JWTs) [73] are typically used as self-contained access tokens, where a JWT has the following components each encoded using the `base64` encoding: a JWT `header`, a JWT `payload` and a signature on the pair (`header`, `payload`). The authorization server sends such a JWT along with other related information (e.g., type, expiry, scope of the token) to the client as JSON claim-value pairs.

The signing algorithm to use in a JWT can be chosen from several options [74] — HS`XXX` (HMAC [75]), RS`XXX` (RSASSA-PKCS1-v1.5 [76]), ES`XXX` (ECDSA [77]) and PS`XXX` (RSASSA-PSS [76]) — with SHA-`XXX` used as the

hashing algorithm for each `XXX` $\in \{256, 384, 512\}$. The chosen algorithm is specified in the JWT `header`. For our threshold JWT implementations, we have considered the RS256 algorithm, where a standard RSA signature from a single authorization server has been replaced by signature-shares from multiple authorization servers. In order to incorporate threshold signature-share generation, we have modified the standard version of the encoding procedure (that also performs signing) for JWTs. We have introduced a procedure at the client side that checks proofs of correctness and combines signature-shares. The client, upon receiving at least k valid signature-shares, combines k such shares into the final signature which is a standard RSA signature (with SHA-256 used as the hashing algorithm). Thus, we can use the standard version of the decoding procedure (that also performs verification of the standard RSA signature on a JWT) at the resource gateway.

We note that, in the threshold setting, each authorization server issues an access token (JWT) along with a signature-share to the client. Thus, in order to ensure that the final JWT signature is done on the same signing input (`header`, `payload`), this input pair generated by all authorization servers for same connection request must be same. As we assume that the instances of the trust algorithm executed by all the servers are same and the databases fed as input to these instances are same, the claim-value pairs in the `payload` of individual JWTs are also same. After receiving the signature-shares on the same signing input (`header`, `payload`) from at least k authorization servers, the client produces the JWT with the final signature to the resource gateway and is granted/denied access to the requested resource accordingly. To sum up, we can say that, by distributing trust among multiple authorization servers using the threshold techniques of `DistriTrust`, the access token generation in OAuth 2.0 can be made more fault-tolerant and hard to compromise while maintaining low-latency authorization.

9. Conclusion

Zero-trust architecture eliminates inherent trust assumptions on the users/devices residing inside the network perimeter of an enterprise. In a zero-trust enterprise network, a policy decision point (PDP) is responsible for validating and granting a user/device an access to the enterprise resource requested. In this work, we have used techniques from threshold cryptography to distribute trust over multiple PDPs. Based on the latency involved in different threshold signature schemes, we have identified a scheme with low latency for a multi-PDP zero-trust network. We have used this scheme to construct `DistriTrust`, a distributed and low-latency access validation framework for a zero-trust enterprise network. Finally, we have discussed the security and performance of `DistriTrust`. Our experimental results show the efficiency of each of the phases – generating signature-shares, combining them into a final signature and verifying the final signature – involved in `DistriTrust`. Among other use cases, we have discussed how the threshold techniques used in `DistriTrust` can also be

employed to collaboratively sign access tokens in OAuth 2.0 in order to make the authorization framework more robust and fault-tolerant.

Acknowledgments

This research is supported by the Agency for Science, Technology and Research (A*STAR) under its IAF-PP (A20F8a0044). The authors would like to thank Sriram Ramachandran, Leonit Zeynalvand and the anonymous reviewers for their insightful comments and suggestions regarding this work.

References

[1] R. S. McCoy, Perimeter security, in: Information Security Management Handbook, Sixth Edition, Auerbach Publications/CRC Press, 2007, pp. 1275–1288.

[2] Palo Alto Networks, Simplify zero trust implementation using a five-step methodology, https://www.paloaltonetworks.com/resources/whitepapers/simplify-zero-trust-implementation-with-a-five-step-methodology (Accessed: July 2021).

[3] P. Dutta, Insider threat: The biggest contributor to cyber attacks, https://www.kratikal.com/blog/insider-threat-the-biggest-contributor-to-cyber-attacks/ (Accessed: July 2021).

[4] S. Ben, Twitter hack: The spotlight that insider threats need, https://beta.darkreading.com/attacks-breaches/twitter-hack-the-spotlight-that-insider-threats-need (Accessed: July 2021).

[5] D. Hein, Managed cloud services: The benefits of outsourcing cloud management, https://solutionsreview.com/cloud-platforms/managed-cloud-services-the-benefits-of-outsourcing-cloud-management/ (Accessed: July 2021).

[6] Forrester Research Inc., Developing a framework to improve critical infrastructure cybersecurity, https://www.nist.gov/system/files/documents/2017/06/05/040813_forrester_research.pdf (Accessed: July 2021).

[7] L. Columbus, IBM's 2018 data breach study shows why we're in a zero trust world now, https://www.forbes.com/sites/louiscolumbus/2018/07/27/ibms-2018-data-breach-study-shows-why-were-in-a-zero-trust-world-now/#6f9168a068ed (Accessed: July 2021).

[8] J. L. Hardcastle, Google brings BeyondCorp zero-trust security to the masses, https://www.sdxcentral.com/articles/news/google-productizes-beyondcorp-zero-trust-network-security/2020/04/ (Accessed: July 2021).

[9] R. Ward, B. Beyer, BeyondCorp: A new approach to enterprise security, ;login: 39 (6) (2014) 6–11.

[10] Microsoft Corporation, Implementing a zero trust security model at Microsoft, https://www.microsoft.com/en-us/itshowcase/implementing-a-zero-trust-security-model-at-microsoft (Accessed: July 2021).

[11] T. Dawoud, Zero trust deployment guide for Microsoft Azure Active Directory, https://www.microsoft.com/security/blog/2020/04/30/zero-trust-deployment-guide-azure-active-directory/ (Accessed: July 2021).

[12] S. Rose, O. Borchert, S. Mitchell, S. Connelly, Zero trust architecture, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf (Accessed: July 2021).

[13] Y. Desmedt, Society and group oriented cryptography: A new concept, in: Advances in Cryptology - CRYPTO, Vol. 293, 1987, pp. 120–127.

[14] Y. Desmedt, Y. Frankel, Threshold cryptosystems, in: Advances in Cryptology - CRYPTO, 1989, pp. 307–315.

[15] Y. Desmedt, Y. Frankel, Shared generation of authenticators and signatures (extended abstract), in: Advances in Cryptology - CRYPTO, 1991, pp. 457–469.

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

[17] A. Shamir, How to share a secret, Communications of the ACM 22 (11) (1979) 612–613.

[18] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, Robust and efficient sharing of RSA functions, in: Advances in Cryptology - CRYPTO, Springer, 1996, pp. 157–172.

[19] T. Rabin, A simplified approach to threshold and proactive RSA, in: Advances in Cryptology - CRYPTO, 1998, pp. 89–104.

[20] S. K. Langford, Threshold DSS signatures without a trusted party, in: Advances in Cryptology - CRYPTO, 1995, pp. 397–409.

[21] V. Shoup, Practical threshold signatures, in: Advances in Cryptology - EUROCRYPT, 2000, pp. 207–220.

[22] T. Rabin, Robust sharing of secrets when the dealer is honest or cheating, Journal of the ACM 41 (6) (1994) 1089–1109.

[23] T. Rabin, M. Ben-Or, Verifiable secret sharing and multiparty protocols with honest majority (extended abstract), in: ACM Symposium on Theory of Computing, STOC, 1989, pp. 73–85.

[24] D. Chaum, H. V. Antwerpen, Undeniable signatures, in: Advances in Cryptology - CRYPTO, 1989, pp. 212–216.

[25] P. Feldman, A practical scheme for non-interactive verifiable secret sharing, in: Annual Symposium on Foundations of Computer Science, FOCS, 1987, pp. 427–437.

[26] S. Miyazaki, K. Sakurai, M. Yung, On threshold RSA-signing with no dealer, in: International Conference on Information Security and Cryptology, ICISC, 1999, pp. 197–207.

[27] D. Chaum, T. P. Pedersen, Wallet databases with observers, in: Advances in Cryptology - CRYPTO, 1992, pp. 89–105.

[28] B. King, Improved methods to perform threshold RSA, in: Advances in Cryptology - ASIACRYPT 2000, 2000, pp. 359–372.

[29] P. Fouque, J. Stern, Fully distributed threshold RSA under standard assumptions, in: Advances in Cryptology - ASIACRYPT, 2001, pp. 310–330.

[30] I. Damgård, M. Koprowski, Practical threshold RSA signatures without a trusted dealer, in: Advances in Cryptology - EUROCRYPT, 2001, pp. 152–165.

[31] I. Damgård, K. Dupont, Efficient threshold RSA signatures with general moduli and no extra assumptions, in: International Workshop on Theory and Practice in Public Key Cryptography, PKC, 2005, pp. 346–361.

[32] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, Robust threshold DSS signatures, in: Advances in Cryptology - EUROCRYPT, 1996, pp. 354–371.

[33] R. Gennaro, S. Goldfeder, A. Narayanan, Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security, in: International Conference on Applied Cryptography and Network Security, ACNS, 2016, pp. 156–174.

[34] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: Advances in Cryptology - EUROCRYPT, 1999, pp. 223–238.

[35] J. Kilian, A note on efficient zero-knowledge proofs and arguments (extended abstract), in: Annual Symposium on Theory of Computing, STOC, 1992, pp. 723–732.

[36] Y. Lindell, A. Nof, Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody, in: ACM Conference on Computer and Communications Security, CCS, 2018, pp. 1837–1854.

[37] T. Chou, C. Orlandi, The simplest protocol for oblivious transfer, in: International Conference on Cryptology and Information Security in Latin America, LATINCRYPT, 2015, pp. 40–58.

[38] M. Keller, E. Orsini, P. Scholl, Actively secure OT extension with optimal overhead, in: Advances in Cryptology - CRYPTO, 2015, pp. 724–741.

[39] J. Doerner, Y. Kondi, E. Lee, A. Shelat, Threshold ECDSA from ECDSA assumptions: The multiparty case, in: IEEE Symposium on Security and Privacy, S&P, 2019, pp. 1051–1066.

[40] R. Canetti, A. Jain, A. Scafuro, Practical UC security with a global random oracle, in: ACM Conference on Computer and Communications Security, CCS, 2014, pp. 597–608.

[41] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, A. Sahai, Threshold cryptosystems from threshold fully homomorphic encryption, in: Advances in Cryptology - CRYPTO, 2018, pp. 565–596.

[42] A. Jain, P. M. R. Rasmussen, A. Sahai, Threshold fully homomorphic encryption, IACR Cryptology ePrint Archive 2017 (2017) 257.

[43] X. Chen, F. Zhang, D. M. Konidala, K. Kim, New ID-based threshold signature scheme from bilinear pairings, in: International Conference on Cryptology in India, INDOCRYPT, 2004, pp. 371–383.

[44] M. J. Fischer, The Legendre and Jacobi symbols, `https://zoo.cs.yale.edu/classes/cs467/2010s/course/handouts/ho07.pdf` (Accessed: July 2021).

[45] Wikipedia, Centralized database, `https://en.wikipedia.org/wiki/Centralized_database` (Accessed: July 2021).

[46] M. Bellare, P. Rogaway, Random oracles are practical: A paradigm for designing efficient protocols, in: ACM Conference on Computer and Communications Security, CCS, 1993, pp. 62–73.

[47] W. Diffie, M. E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (6) (1976) 644–654.

[48] IETF, The Transport Layer Security (TLS) protocol version 1.3, `https://tools.ietf.org/html/rfc8446` (Accessed: July 2021).

[49] IETF, An interface and algorithms for authenticated encryption, `https://tools.ietf.org/html/rfc5116` (Accessed: July 2021).

[50] I. E. Leonard, Notes on the Euclidean algorithm, http://www.math.ualberta.ca/~isaac/math324/s06/euclidean.pdf (Accessed: July 2021).

[51] Wikipedia, Lagrange polynomial, https://en.wikipedia.org/wiki/Lagrange_polynomial (Accessed: July 2021).

[52] GMP, The GNU Multiple Precision Arithmetic Library, https://gmplib.org/ (Accessed: July 2021).

[53] OpenSSL, Cryptography and SSL/TLS toolkit, https://www.openssl.org/ (Accessed: July 2021).

[54] OverOps, Configure LDAP for on-premises deployments, https://doc.overops.com/docs/configuring-ldap-for-on-premises-deployments (Accessed: July 2021).

[55] Microsoft, Azure active directory, https://azure.microsoft.com/en-us/services/active-directory/ (Accessed: July 2021).

[56] DigiCert, Enterprise PKI manager, https://www.digicert.com/pki/enterprise-pki-manager (Accessed: July 2021).

[57] GlobalSign, Managed PKI platform, https://www.globalsign.com/en-sg/managed-pki (Accessed: July 2021).

[58] Entrust, Public key infrastructure (PKI), https://www.entrust.com/digital-security/certificate-solutions/products/pki (Accessed: July 2021).

[59] Apple Inc., Apple PKI, https://www.apple.com/certificateauthority/ (Accessed: July 2021).

[60] Google, Google trust services, https://pki.goog/ (Accessed: July 2021).

[61] Open Policy Agent, Policy-based control for cloud native environments, https://www.openpolicyagent.org/ (Accessed: July 2021).

[62] Casbin, Casbin overview, https://casbin.org/docs/en/overview (Accessed: July 2021).

[63] P. Samarati, S. D. C. di Vimercati, Access control: Policies, models, and mechanisms, in: Foundations of Security Analysis and Design, FOSAD, Vol. 2171 of Lecture Notes in Computer Science, 2000, pp. 137–196.

[64] Microsoft, Always on availability groups: A high-availability and disaster-recovery solution, https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/always-on-availability-groups-sql-server?view=sql-server-ver15 (Accessed: July 2021).

[65] PostgreSQL, High availability, load balancing, and replication, https://www.postgresql.org/docs/9.5/high-availability.html (Accessed: July 2021).

[66] Oracle Corporation, Oracle database high availability solutions, https://docs.oracle.com/cd/B19306_01/server.102/b14210/hafeatures.htm (Accessed: July 2021).

[67] OpenLDAP Foundation, OpenLDAP, https://www.openldap.org/ (Accessed: July 2021).

[68] OpenLDAP Foundation, Replication, https://www.openldap.org/doc/admin24/replication.html (Accessed: July 2021).

[69] Oracle Corporation, Replication, https://dev.mysql.com/doc/refman/8.0/en/replication.html (Accessed: July 2021).

[70] IBM Corporation, iBase database replication, https://www.ibm.com/docs/en/i2-ibase/9.0.2?topic=center-ibase-database-replication (Accessed: July 2021).

[71] L. T. A. N. Brandão, N. Mouha, A. Vassilev, Threshold schemes for cryptographic primitives: Challenges and opportunities in standardization and validation of threshold cryptography, https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8214.pdf (Accessed: July 2021).

[72] IETF, The OAuth 2.0 authorization framework, https://tools.ietf.org/html/rfc6749 (Accessed: July 2021).

[73] IETF, JSON Web Token (JWT), https://tools.ietf.org/html/rfc7519 (Accessed: July 2021).

[74] IETF, JSON Web Algorithms (JWA), https://tools.ietf.org/html/rfc7518 (Accessed: July 2021).

[75] IETF, HMAC: Keyed-hashing for message authentication, https://tools.ietf.org/html/rfc2104 (Accessed: July 2021).

[76] IETF, PKCS #1: RSA cryptography specifications version 2.2, https://tools.ietf.org/html/rfc8017 (Accessed: July 2021).

[77] D. Johnson, A. Menezes, S. Vanstone, The elliptic curve digital signature algorithm (ECDSA), International Journal of Information Security 1 (1) (2001) 36–63.