# ARPA Network Whitepaper

Penrose

September 30, 2022

**Abstract**

ARPA Network provides an efficient, permissionless threshold signature service for blockchains. At its core, the ARPA Network contains a threshold BLS signature that meets the need for decentralization, non-interactivity, verifiability, and high availability. The system is designed and implemented to work with blockchains and provide threshold signature capability for users. Nodes of the ARPA Network are grouped to handle computation tasks in parallel. We utilize a smart-contract-capable blockchain as a bulletin to manage dynamic global states and coordinate multiple groups. It is expected that developers can build applications like secure key management, cross-chain messaging, and quorum approval on our network. Finally, Randcast is proposed as a distributed random number generator as a use case of the ARPA network.

## 1 Introduction

Blockchain technology has proven itself to be a significant part of the global internet infrastructure. During the past decade, academic and industrial experts have extensively improved its privacy, scalability, and interoperability. An effective means to enhance the blockchain is by verifiably outsourcing computation and storage from on-chain to off-chain through threshold cryptography schemes. These schemes offer trust through their provable security and decentralization, which matches the distributed and trustless nature of the blockchain. At the same time, as a bulletin board, the blockchain can serve as a reliable broadcast channel for these cryptographic algorithms. Therefore, combining threshold cryptography with blockchains can simultaneously strengthen blockchains and facilitate the implementation of threshold cryptography.

In this paper, we propose ARPA Network, a permissionless distributed network, to provide essential threshold signature capability for blockchains. Threshold signature is a protocol that allows for processing signature-related functions among a group of nodes through multi-party computation (MPC). It can help with keeping secrets distributed, reaching consensus by majority vote, or hiding the identity of signers. With the aid of the ARPA Network, one can build applications involving secure key management, anonymous transaction, cross-chain messaging, quorum approval, distributed randomness generation, etc. Like blockchain, the trustworthiness of the ARPA Network comes from its distribution across independent nodes. These nodes are bundled into groups, with each group capable of executing the threshold Boneh–Lynn–Shacham (BLS) signature scheme. Thanks to the aggregatability of BLS signatures and the node management mechanism we apply, the ARPA Network has the following features.

- Decentralization: ARPA nodes provide threshold signature service in a decentralized manner. Trust is distributed to multiple entities located in different regions running individual nodes. This manner offers better tamper protection at the physical level.
- Flexibility: ARPA protocol supports a variety of signature applications. Users are allowed to customize their signature policy, import or export secrets, and choose the security level.

- Verifiability: ARPA signature scheme allows users to trivially verify their signatures. By virtue of the underlying cryptographic primitive, the signatures cannot be forged or manipulated.
- Non-interactivity: ARPA protocol avoids heavy synchronous communication in the signature generation phase. A non-interactive procedure guarantees reliable service status and flexible network topology.
- High availability: ARPA signature service keeps a high availability thanks to its decentralization and non-interactivity. As the network grows, the downtime will reduce accordingly.
- Multi-Chain Support: ARPA network is designed to support multiple chains, allowing developers from diverse ecosystems to leverage ARPA when building their applications. In addition, our underlying threshold signature network will keep consistency between the different data replicas.

In the remainder of this paper, we first overview the existing threshold signature schemes and point out the reason why threshold BLS signature is suitable for distributed systems in section 2. Then section 3 lists out presuppositions and the building blocks of our design. A walk-through of our implemented protocol is given in section 4, including the cryptographic part and the node management mechanism. Section 5 presents the system design by showing the high-level architecture of the network. Finally, section 6 demonstrates Randcast, a distributed pseudo-random number generator, as a use case of the ARPA Network.

## 2 Background and Related Works

Threshold signature schemes have been researched and extensively applied in the last few years. Many classical signature algorithms have been thresholdized, including BLS, ECDSA [10], EdDSA [17], RSA [8]. However, Some threshold signature schemes are not well-suitable for use in distributed systems. For example, threshold ECDSA takes 3 to 13 synchronous communication rounds depending on the construction methods used [1]. In real-world situations, transmission latency determines the performance of the MPC protocols. Multiple rounds of communications will undoubtedly affect the average speed of threshold signature generation. Moreover, a failed node may even cause the protocol to abort. Fortunately, by utilizing a bilinear pairing, the BLS signature may help to mitigate this problem.

A pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, which has bilinearity,

$$\forall a, b \in \mathbb{Z}_r^*, P \in \mathbb{G}_1, Q \in \mathbb{G}_2, e(aP, bQ) = e(P, Q)^{ab}. \tag{1}$$

The BLS signature scheme is built upon the pairing while its signatures and public keys lie in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Without loss of generality, we represent signatures by elements in $\mathbb{G}_1$ to achieve a more compact signature. It can be seen from equation 1 that there exists a homomorphic mapping between the secret keys and the signatures. This means specific computations on generated signatures imply corresponding computations on secret keys. This makes the asynchronous threshold signature scheme possible.

Threshold BLS signatures have also been studied by some authors. Some of them consider multi-signatures. They can be seen as a special threshold signature scheme that requires all parties to participate in signature generation rather than a part of them. Compared with a multi-signature scheme built on other signatures, BLS signatures can be aggregated publicly by a simple multiplication, even when original signers are offline. e.g., a BLS multi-signature for Bitcoin is described in Boneh, Drijvers, and Neven [2].

As for the $t$-out-of-$n$ threshold BLS signature schemes, the main difference among them is the choice of distributed key generation (DKG) sub-protocol. Generally speaking, a DKG protocol

generates public and private key shares for nodes without computing the private key. It can be seen as several parallel executions of verifiable secret sharing (VSS), which is the cornerstone of most MPC protocols. DKG protocols have different communication settings, synchrony and asynchrony. An asynchronous verifiable secret sharing scheme is intractable to design, but there are still several teams researching this subject. Unconditionally secure AVSS has a communication complexity of $\Omega(n^5)$, making it unrealistic to use. Compromising the unconditional security assumption, several schemes provide a more practical performance but are still generally ineffective [13].

Synchronous VSS is more practical to design and deploy. Galindo [9] implements its DKG by Pedersen's VSS [16]. The Keep Network [18] uses DKG proposed by Gennaro [11] to generate keypairs. These DKGs are all based on the Joint-Feldman Distributed Key Generation (JF-DKG) mentioned by Pedersen [15]. It has been discussed in Gennaro et al. [11] that the public key generated by JF-DKG could be biased by a static adversary. However, the hardness of the elliptic curve discrete-log problem (ECDLP) for the public key will not be weakened. It is sufficiently secure for threshold BLS signatures. Considering the simplicity and efficiency of the protocol, we will deploy JF-DKG to generate key shares for nodes in our system. Other DKG variants will be integrated in the future to support different scenarios.

# 3 Presuppositions

While building an underlying threshold signature service for the blockchain, we should first clarify the security and communication model. The following outlined assumptions are highly relevant to the characteristics of the blockchain, where the system is distributed and permissionless while the participants are economically rational but potentially malicious.

## 3.1 Security Assumption

We assume that the adversary is a static, malicious, honest-majority adversary. The attacker can corrupt up to $t$ of the $n$ parties in the network at one time, where $t < n/2$. The corrupted parties may divert from the prescribed protocol in any way. Considering the malicious behaviors, honest-majority is the best achievable threshold for protocols that provide both secrecy and robustness [11]. The static adversary would choose the corrupted parties far ahead of time, which means getting control of a particular party is non-trivial. But it is possible that the corrupted parties may vary for a long time. Therefore, a key rotation or refreshment scheme is introduced to cope with the long-term key exposure.

## 3.2 Communication Model

The distributed signature network is composed of a set of $n$ parties $P_1, \cdots, P_n$ that are connected by a complete network of private point-to-point channels. In addition, the parties have access to a dedicated broadcast channel. Several works have researched the threshold signature without a preset reliable broadcast [12, 13, 6]. They assimilate various secret sharing schemes into reliable broadcast or consensus protocols. AVSS [6] merges a bivariate polynomial into Bracha's reliable broadcast [5]. Gennaro [11] leverages Byzantine fault tolerant (BFT) protocol [7] to build the DKG for the Internet. It can be concluded from these articles that constructing a reliable broadcast channel is almost equivalent to reaching a consensus among nodes. Considering there are kinds of blockchains and smart contracts that help us broadcast and record messages publicly, it is reasonable to offload this part of a protocol to blockchains.

### 3.3 Synchrony Assumption

Based on the block generation mechanism, we may assume a partially synchronous communication model: the protocol proceeds in synchronized rounds, and messages are received by their recipients within some specified time-bound. The block time of a blockchain that guarantees liveness can be used as the synchronized clock in the threshold signature. The drawback is that a typical block time is several orders of magnitude longer than an Internet transmission. The partial synchrony opens up an avenue of attack where an adversary can observe the messages of the uncorrupted parties, then decide on his action during each round of the protocol, and still get his messages delivered to the rest of the parties on time. In the worst case, the adversary may speak last in every communication round to exploit back-running [11].

## 4 ARPA Threshold Signature Protocol

Based on the assumptions made previously, ARPA Network builds up the protocol by incorporating Joint-Feldman Distributed Key Generation into standardized BLS signature [3] on curve BLS 12-381 [4]. Furthermore, ARPA Network combines the threshold BLS signature with the blockchain. The threshold BLS signature is responsible for generating a decentralized tamper-proof signature, and the blockchain provides a reliable broadcast channel as well as coordinating functionality. Procedures for node registration, secession, grouping, as well as other node management mechanisms are also defined.

At a high level, the network initializes by allowing nodes to stake and join. These nodes will then be divided into groups and complete the DKG process. After the networks has started, the user can request services by sending a transaction to a specific smart contract on their blockchain. The task will then be forwarded to the ARPA network and get assigned to one of the groups of network nodes. This group of nodes will then collectively generate a signature on the message provided and return it back to the blockchain.

### 4.1 Distributed Key Generation

As an essential component of threshold cryptosystems, a distributed key generation protocol is responsible for generating private and public keys of participants. The underlying secret sharing scheme used in the DKG decides the relationship of key shares held by each participant. This also fundamentally determines the key management policy and the signature generation algorithm used in the threshold signature scheme. Our deployed JF-DKG is described in algorithm 1.

### 4.2 Threshold BLS Signature

BLS signature is first proposed as a short signature scheme where the signatures consist only of a single group element. ElGamal type schemes such as digital signature algorithm (DSA) and elliptic curve digital signature algorithm (ECDSA) produce signatures comprised of a pair of integers. Therefore, BLS signatures are about half the length of those produced by other widely used schemes at the same security level [14]. BLS signatures utilize a bilinear pairing which offers several interesting features. Firstly, BLS signatures are deterministic given a particular message and a keypair, unlike ECDSA, which requires a fresh random value for each signing. This prevents signers from biasing results by repeated signing attempts. Secondly, BLS signatures are aggregatable, which means specific computations on generated signatures are possible. This makes a difference between the threshold BLS signature and other threshold signatures. Multiple rounds of synchronous communication are unnecessary for combining partial BLS signatures. The BLS signature scheme consists of the following sub-procedures.

**Algorithm 1** Joint-Feldman Distributed Key Generation [11]

**Require:** Adversary threshold $t$, group size $n$, an elliptic curve $\mathbb{G}_2$ with a prime order $r$ and a generator $g_2$.

**Ensure:** Shamir secret shares $sk_i$ for party $P_i$ respectively.

1: Each party $P_i$ chooses a $f_i(z) = \sum_{k=0}^{t} a_{ik} z^k$, where $a_{ik} \in_R \mathbb{Z}_r^*$, broadcasts commitment $C_{ik} = g_2^{a_{ik}}$ for $k \in [0, t]$. Each party $P_i$ computes the shares $s_{ij} = f_i(j) \bmod r$ for $j \in [1, n]$ and sends $s_{ij}$ to party $P_j$ secretly.

2: Each party $P_j$ verifies the shares his received $g_2^{s_{ij}} = \prod_{k=0}^{t} (C_{ik})^{j^k}$ for $i \in [1, n]$. If the check for an index $i$ fails, $P_j$ raises a complaint against party $P_i$

3: Party $P_i$ reveals the shares corresponding to raised complaints. Each party $P_j$ checks the validity between complaints and equations. Any failed party will be contained in a local disqualified set of each party. $QUAL$ is defined as the set of non-disqualified parties.

4: The public key $y$ is computed as $pk = \prod_{i \in QUAL} C_{i0}$. The secret shared value $sk$ itself is not computed by any party, but it is equal to $sk = \sum_{i \in QUAL} a_{i0} \bmod r$. Each party $P_j$ sets his share of the secret as $sk_j = \sum_{i \in QUAL} s_{ij} \bmod r$, and the corresponding public key share as $pk_j = \prod_{i \in QUAL} g_2^{s_{ij}} = \prod_{i \in QUAL} \prod_{k=0}^{t} (C_{ik})^{j^k}$

**Key Generation** To generate a key pair, a signer first chooses a random secret $sk \in_R \mathbb{Z}_r^*$ and computes the corresponding public key as $pk = g_2^{sk} \in \mathbb{G}_2$.

**Signing** To compute a BLS signature $\sigma$ on an arbitrary message $m$, the signer computes $\sigma = sk \times H(m) \in \mathbb{G}_1$. $H(m)$ is a hash-to-curve function that maps an arbitrary bit string to an element in $\mathbb{G}_1$.

**Verification** To verify the validity of a BLS signature $\sigma$ on a given message $m$, the verifier checks if $(g_2, pk, H(m), \sigma)$ is a Diffie-Hellman quadruple, which will be the case that $e(\sigma, g_2) = e(H(m), pk)$ holds.

A threshold signature scheme is computing a signature among a group of independent nodes MPC. Considering the different processes of various signature schemes, the construction of their threshold versions will be very dissimilar. For example, the threshold ECDSA involves homomorphic multiplication of encrypted messages resulting in a complex sub-protocol called Multiplicative to Additive (MtA). It takes several rounds of synchronous communications to process. As for the BLS signature, thanks to the homomorphic property, the threshold BLS signature is neat and clear. Its sub-procedures are as follows.

**Distributed Key Generation** The $n$ parties jointly execute Algorithm 1 to generate a group public key $pk \in \mathbb{G}_2$, a virtual private key $sk \in \mathbb{Z}_r^*$, and their shares $pk_i$, $sk_i$.

**Partial Signature Generation** To sign a message $m$, each party $P_i$ individually signs the partial BLS signature $\sigma_i = sk_i \times H(m)$ by his private key share $sk_i$.

**Partial Signature Verification** To validate a partial BLS signature, one uses the corresponding public key share to check if $e(H(m), pk_i) = e(\sigma_i . g_2)$ holds.

**Signature Reconstruction** To reconstruct the BLS signature $\sigma$ on $m$, one has to collect $t+1$ valid and independent partial signatures, then compute

$$\sigma = \prod_{k=0}^{t} \sigma_{i_k} \prod_{m=0, m \neq k}^{t} \frac{i_m}{i_m - i_k}, \quad i_k, i_m \in [1, n].$$

**Verification** To verify the validity of a BLS signature $\sigma$ on a given message $m$, the verifier checks if $(g_2, pk, H(m), \sigma)$ is a Diffie-Hellman quadruple, which will be the case that $e(\sigma, g_2) = e(H(m), pk)$ holds.

Thanks to the properties of Feldman's VSS scheme, the generated signature $\sigma$ is identical no matter which partial signatures are chosen during reconstruction. Meanwhile, the determinism of BLS signature also guarantees the immutability of the threshold signature no matter what kinds of (computationally bounded) attacks are conducted. We now have a decentralized, verifiable, tamper-proof threshold BLS signature scheme.

# 5 ARPA Network System Design

The ARPA Network is designed and implemented as a distributed system to process the threshold BLS signature scheme, as shown in Figure 1. The network is compatible with any smart-contract-capable blockchain. The only requirement to support a new blockchain is the creation of a smart contract to integrate the ARPA Network with the target blockchain. Logically, the ARPA Network has two layers: the service layer and the provision layer. The service layer conducts the above-mentioned threshold BLS signature scheme, while the provision layer manages the nodes and reliably broadcasts information through blockchain.

When integrating with a blockchain, The ARPA Network needs a "controller" smart contract to manage its dynamic global states. On a high level, these dynamic global states include node information, group information, and all BLS signature tasks. Each unregistered node directly interacts with the "controller" to register themselves into the ARPA Network and discover the information needed to communicate with other registered nodes.

For higher throughput and better service availability, the nodes in the ARPA network are split into multiple groups to handle BLS signature tasks in parallel. The "controller" is also responsible for initiating the grouping of the nodes and storing the group information. The grouping process is essentially a DKG process. A "coordinator" smart contract is deployed ad-hoc to coordinate a subset of nodes through the different phases of the DKG process, then submits the proof of DKG completion to the "controller" for verification.

The "controller" also acts as an API endpoint which provides BLS signature services to other DApp clients. The DApp client requests the BLS signature by calling their smart contract extending our "consumer" smart contract, which in turn calls the underlying "controller" APIs. The "controller" assigns the BLS signature task to a specific group. Each grouped node monitors the BLS signature task event emitted by the "controller" and starts a BLS signature task if it belongs to the assigned group, then submits the signature to the "controller" upon completion. The "controller" then returns the results to the caller DApps. A backup mechanism is also in place if the assigned group fails to fulfill the request within a reasonable time.

# 6 Randcast

The ARPA Network can be leveraged to build a variety of applications. A distributed random number generator (DRNG) is one possible use case. Like the physical world, where the whole universe is built upon random motions of molecules, random numbers are ubiquitous and essential in cyberspace. A trustworthy and reliable pseudo-random number generator is a cornerstone to both blockchain infrastructures as well as the applications built upon it. In this section, we present Randcast, a randomness generation application of the ARPA Network.

An easy approach to generating a random number is through a trusted third party. However, centralized randomness generation suffers from trust degradation with a concern for the backdoors.

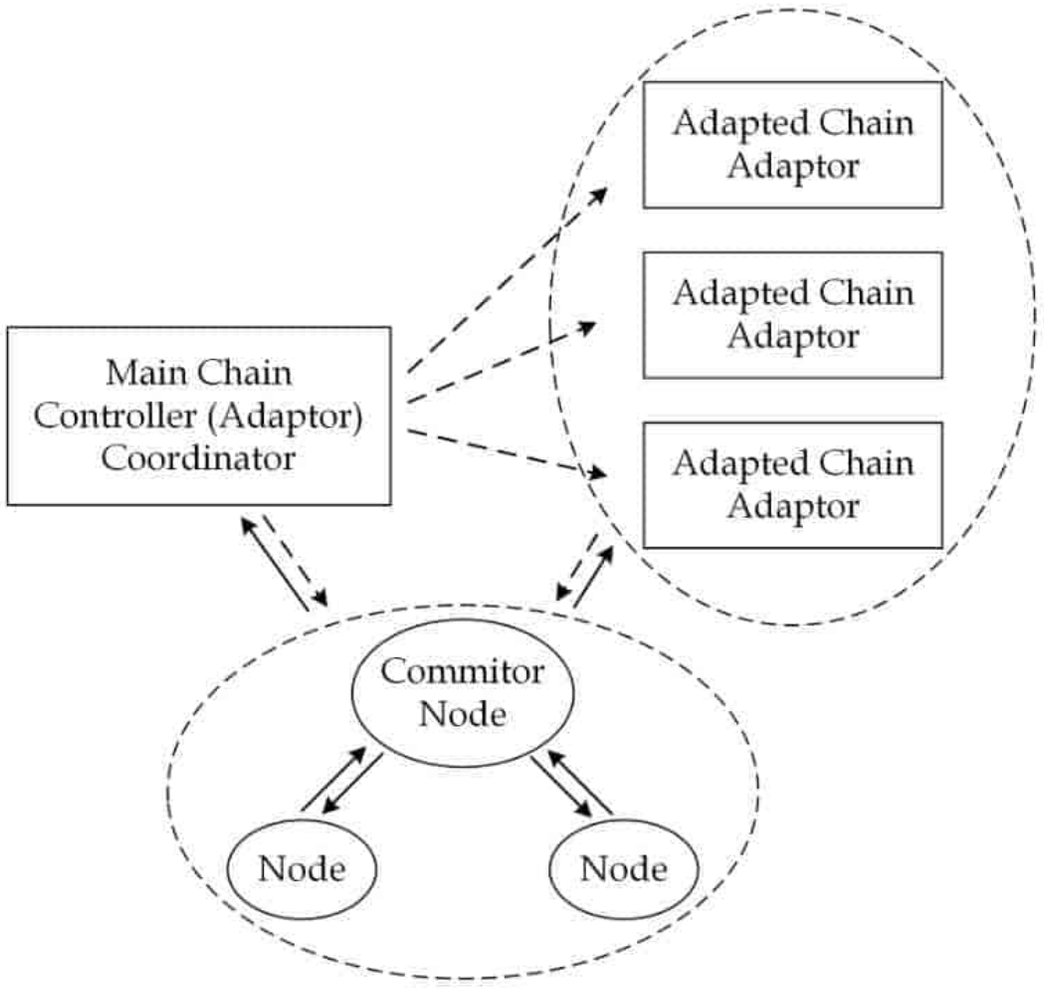


Figure 1: A high-level architectural view of the ARPA Network (this architecture diagram may need a change)

When used in a distributed system, a DRNG is desirable, and the threshold BLS signature is a cryptographic primitive to build one. Randcast leverages the ARPA Network to provide randomness service. Once the network receives a seed for a random number generation task, one group is called to generate a signature on the seed. The signature can then be used as a deterministic, verifiable, unforgeable random number. Inheriting from the threshold signature network, Randcast has the following features.

- Decentralization: Randcast produces random numbers in a decentralized manner. Entropy is gathered from multiple entities located in different regions running individual nodes. This manner offers unpredictability and fairness at the physical level.
- Uniqueness: Randcast generates random numbers depending only on request messages and node secrets. Given certain signing group and user input, the randomness is unique and tamper-proof, which mitigates corruption inside the network.
- Verifiability: Randcast allows everyone to check the validity of the random number. By virtue of underlying cryptographic primitive, The random number is unlikely to be forged or manipulated.
- Non-interactive: Randcast avoids heavy synchronous communication in the random generation phase. A non-interactive procedure guarantees a better service status.
- High availability: Randcast has a high availability thanks to its decentralization and non-interactivity. As the network grows, downtime will reduce accordingly.
- Multi-Chain Support: Randcast is designed to adapt to multiple chains, allowing developers from multiple ecosystems to cast "randomness" spells. Our underlying threshold signature network will keep consistency between the different data replicas.

# A BLS 12-381 specification

BLS12-381 is a specific curve of a pairing-friendly curve family. It has an embedded degree as 12 and a 381-bit field prime. The public parameters are outlined as follows[4].

$$
\begin{aligned}
u = \texttt{-0x} \quad & \texttt{d2010000 00010000} \\
k = \quad & 12 \\
q = \ \texttt{0x} \quad & \texttt{1a0111ea 397fe69a 4b1ba7b6 434bacd7 64774b84 f38512bf 6730d2a0} \\
& \texttt{f6b0f624 1eabfffe b153ffff b9feffff ffffaaab} \\
r = \ \texttt{0x} \quad & \texttt{73eda753 299d7d48 3339d808 09a1d805 53bda402 fffe5bfe ffffffff} \\
& \texttt{00000001} \\
E(\mathbb{F}_q) := \quad & y^2 = x^3 + 4 \\
\mathbb{F}_{q^2} := \quad & \mathbb{F}_q[i]/(x^2 + 1) \\
E'(\mathbb{F}_{q^2}) := \quad & y^2 = x^3 + 4(i + 1)
\end{aligned}
$$

# References


[1] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. A survey of ecdsa threshold signing. *Cryptology ePrint Archive*, 2020.

[2] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 435–464. Springer, 2018.

[3] Dan Boneh, Sergey Gorbunov, Riad S. Wahby, Hoeteck Wee, Christopher A. Wood, and Zhenfei Zhang. BLS Signatures. Internet-Draft draft-irtf-cfrg-bls-signature-05, Internet Engineering Task Force, June 2022. Work in Progress.

[4] Sean Bowe. BLS 12-381: New zk-SNARK Elliptic Curve Construction. `https://electriccoin.co/blog/new-snark-curve/`, 2017. [Online; accessed 31-May-2022].

[5] Gabriel Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162, 1984.

[6] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 88–97, 2002.

[7] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[8] Ivan Damgård and Maciej Koprowski. Practical threshold rsa signatures without a trusted dealer. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 152–165. Springer, 2001.

[9] David Galindo, Jia Liu, Mihair Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 88–102. IEEE, 2021.

[10] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1179–1194, 2018.

[11] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[12] Aniket Kate and Ian Goldberg. Distributed key generation for the internet. In *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 119–128. IEEE, 2009.

[13] Aniket Kate, Yizhou Huang, and Ian Goldberg. Distributed key generation in the wild. *Cryptology ePrint Archive*, 2012.

[14] Alfred Menezes. An introduction to pairing-based cryptography. *Recent trends in cryptography*, 477:47–65, 2009.

[15] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[16] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 522–526. Springer, 1991.

[17] Douglas R Stinson and Reto Strobl. Provably secure distributed schnorr signatures and a (t, n) threshold scheme for implicit certificates. In *Australasian Conference on Information Security and Privacy*, pages 417–434. Springer, 2001.

[18] The Keep Network Team. The Keep Random Beacon: An Implementation of a Threshold Relay. `https://docs.keep.network/random-beacon/`, 2022. [Online; accessed 31-Aug-2022].