Polka.Domain

Take Ownership of Your Digital Identity and Assets

March 2021
Version 1.0.0

Abstract

Cryptocurrency wallet addresses are a hassle to deal with when sending assets to others. These hexadecimal addresses are long sequences of letters and numbers with room for mistakes and far from being user-friendly. Moreover, disclosing a public address traces all past and future transactions, which poses a safety and privacy issue. In this whitepaper, Polka.Domain solves this challenge by providing a digital representation on the blockchain network: 1) Provide a decentralized domain naming service secured by the Polkadot network. 2) Integrate crypto addresses to a decentralized domain (possibly with identities) while preserving user privacy. 3) Accord end-users the ownership of their data and resources. We contribute to the grand scene of Web3.0 by 1) offering cross-chain compatibility and interoperability for cross-chain multi-assets transactions and 2) building a decentralized marketplace for domains and NFT for all.

Contents

1 Introduction

Many fail to realize that DNS is already decentralized, with the exception of a single, critical component, of which trust is centralized: the root zone, or simply, a collection of top level domains (TLDs). And this trust anchor is kept by a small federation of authoritative bodies, where ICANN (Internet Corporation for Assigned Names and Numbers) is currently the ultimate authority. ICANN is an a multistakeholder group and nonprofit organization responsible for coordinating the maintenance and procedures of several databases related to the namespaces and numerical spaces of the Internet, ensuring the network's stable and secure operation. Polka.Domain is a project that intends to disintermediate the role of ICANN and, in its place, applies a decentralized blockchain using threshold signatures to operate this integral piece of infrastructure to maintain the very security of the Internet itself.

Online social networks, especially microblogging networks, are getting more and more important in our daily life. Microblogging networks to spread information, ideas, and influence via social links has been a trending practice. Individuals at central or critical positions in the microblogging networks are expected to play a more vital role in spreading information. An individual may have different microblogging networks for different purposes (e.g., Youtube, Instagram, Twitter, LinkedIn, Facebook). However, having centralized microblogging networks is vulnerable to single-point-of-failure. Due to centralization, users may be blocked to use their own domains which lead to lose the democratization of world. Decentralized microblogging networks is therefore getting more attention after the invention of blockchains. By Polka.Domain, all social networks can be managed by a single domain and will only be controlled by the owner with an unstoppable feature using decentralized trusted system (e.g., Youtube, Instragram, Twitter, LinkedIn, Facebook all will be in one page of a domain owner and with possibly payment system in case of sharing valuable data with the clients/users).

COVID19 shows us that both remote working and self-employment through independent contracting can offer a highly flexible career option that allows people to improve their talents and skills. Performing multiple projects may also provide variety, security and, possibly, additional income. Therefore, microblogging networks will be more important to show individuals' capabilities and share their valuable experience or data. The current situation is not working well due to lack of incentivization or payment channel for both content creator as well as clients/users. In fact, there are more than 2.5 quintillion bytes of data created each day at our current pace, but will be expected to accelerate with the growth of the Internet of Things (IoT). Polka.Domain fills this gap and eliminates the oligopoly of all centralized systems, namely all microblogging networks will be on a single domain with a fully controlled by the owner.

By Polka.Domain, everyone can create his/her own domain (with an extension .pd) and a wallet, and then they can use it to present their social platforms (e.g., Youtube, Instagram, their online storefronts) with processing payments over the underlying blockchain platform through that domain. Hence, Polka.Domain presents everyone to own one web-page of themselves. More importantly, by Polka.Domain, these domains will also be used to store cryptocurrency addresses in a privacy

manner. Namely, anyone sending a cryptocurrency to someone requires the use (i.e., by copy-paste) of a long randomized string. This is certainly one of the main barrier to make it widespread. On the other hand, if a cryptocurrency address of someone with its identity is made public then anyone can trace past and future transactions. This is also certainly not accepted due to privacy reasons. Therefore, Polka.Domain solves a long-standing problem of usable and privacy preserving wallet addresses where it integrates all cryptocurrency addresses of the user with his/her identity in a privacy manner using hierarchical deterministic wallets (BIP32).

Furthermore, Polka.Domain is built on and secured by the Polkadot network. Unlike Bitcoin and Ethereum, Polkadot enables any data to be sent between any blockchain, in turn allowing for multiple transactions to be processed in parallel under shared security, providing a strong backbone for Web3 applications with enhanced usability. Our decentralized domains are represented as tokenized assets in the form of Non-Fungible Tokens. They are unique, not interchangeable, collectible, and can also be traded and circulated as NFT at the same time. We want to enhance our user experience by building a decentralized one-step domain exchange and auction platform. Get ready to buy, sell and trade your domain with us!

2 Usable Key Asset Management Through Polka.Domain

Polka.Domain aims to be the International Cryptocurrency Account Domain (like IBAN in conventional systems) for the blockchain space. The reliability and trust are achieved through threshold signatures to prevent single-point-of-failure. Polka.Domain has been designed on Polkadot blockchain for domains of which keys for the wallet addresses are integrated in a privacy manner. In summary, the key asset management is performed on Polka.Domain which has the following features:

1. **Scalable and Efficient Decentralized Chain:**Polka.Domain nodes will provide an efficient and scalable consensus mechanism which supports hundreds of nodes with thousands of transactions per seconds.

2. **Distributed Trust Certification:** Providing a decentralized trust mechanism removing a single-point-of-failure by means of threshold EdDSA (with the signature aggregation property). This will be provided by means of dynamic threshold signature mechanism for adding/removing signatories to add or remove the signatories in a flexible manner without changing the overall public key.

3. **Privacy-Preserving Digital Asset Key Management:** Integrate cryptowallets to a domain (possibly with identities) while providing privacy by means of secure multiparty computation (SMPC).

4. **Cross-chain, Multi-assets Transactions:** Polka.Domain is built on substrate offering cross-chain compatibility and interoperability backed by Polkadot. Multi-blockchain functionality moves tokens flawlessly across various networks with bridges connecting non-Polkadot chains like Bitcoin and Ethereum to Polkadot.

5. **Domain Marketplace:** Polka.Domain marketplace allows you to buy, sell and exchange domains through auction or fixed swaps.

6. **Register and Launch Censorship Resistant Websites:** There's no authority with the power to enforce censorship on Polka.Domain names. The domains are stored in your wallet, protected by their private key. Only you, have full control, can take down or deny access to your domain.

3 Preliminaries

3.1 Hash Functions

We denote $H(.)$ for the conventional strong collision resistant hashing process as:

$$H : \{0,1\}^* \to \{0,1\}^f, \tag{1}$$

where f is a fixed length of output (e.g., if H is SHA-256 based, then $f = 256$).

3.2 ECDSA Signing Algorithm

Cryptowallets on a very high level are applications to control access to money, manage keys and addresses, track balance, create, and signed transactions. The are actually data structure used to store and manage user's keys. They never contain coins but keys to unlock the coins, namely users sign transactions with the keys, therefore proving that they indeed own the transaction outputs.

ECDSA (e.g., Bitcoin, Ethereum) and EdDSA (e.g., Cardano, NANO, Stellar Lumens, WAVES, Libra) are the most widely used digital signatures used in the blockchain space. It consists of a public and private key pair to control access to crypto funds. In particular, public keys are used to receive funds while private keys are used to digitally sign transactions. In order to spend the funds, the signature can be validated against the public key without revealing the private key.

An elliptic curve (in short affine Weierstrass form) E over a finite field $\mathbb{F}_q$, $q = p^n$ a prime power, (e.g. if $n = 1$ then $\mathbb{F}_q = \mathbb{Z}/p\mathbb{Z}$), is the set of solutions of an equation:

$$E : y^2 = x^3 + Ax + B \text{ where } A, B \in \mathbb{F}_q, \tag{2}$$

that is

$$E(\mathbb{F}_q) := \{(x, y) \in \mathbb{F}_q \times \mathbb{F}_q | y^2 = x^3 + Ax + B\}). \tag{3}$$

The Elliptic Curve Discrete Logarithm Problem (ECDLP) is formulated in the multiplicative group $(\mathbb{F}_p^*, .)$, namely: given $P, Q \in \mathbb{F}_p^*$, find $k \in \mathbb{Z}_p^*$ such that $Q = k.P \bmod p$.

The ECDSA signing algorithm uses an integer a as the private key. Let us quickly outline how it works: We suppose that a user U wants to sign a message m to authorize a payment to a merchant M. Given (public) is an elliptic curve $\mathbb{E}/\mathbb{F}_q$ over a finite field $\mathbb{F}_q$, a base point $P \in \mathbb{E}(\mathbb{F}_q)$ (the base

of our logarithm) of order r, and a hash function H (its values will be considered modulo r) which is preimage and collision resistant.

- Private key: U chooses a secrete integer a mod r, i.e. $a \in \mathbb{Z}_r^*$.
- Public key: U computes $Q = a.P$ and publishes Q. (Note that the address of the user U is derived from Q for Bitcoin).

Then the signing algorithm is as follows:

- **Signing:** U chooses a random (each time a different one!) number $1 < k < r$ and computes $R = (x.R, y.R) = k.P \in \mathbb{E}(\mathbb{F}_q)$. Then U computes $s = k^{-1}(H(m) + a.R) \in \mathbb{Z}_r$ (well-defined). U's signature is then $(x.R, s)$.
- **Verification:** M computes $u_1 = s^{-1}H(m)$ and $u_2 = s^{-1}R \in \mathbb{Z}_r$, and then $V = (xV, yV) = u_1P + u_2Q$ on E. (b) M verifies the signature of user U by checking if $xR = xV \in \mathbb{Z}_p$ holds.

3.3 Edwards-Curve Digital Signature Algorithm (EdDSA)

EdDSA is currently used by Cardano, NANO, Stellar Lumens, WAVES, and Libra, and will be supported by many more blockchains. Compared to ECDSA's signing and verification steps, EdDSA is simpler and easier to understand and to implement. Both signature algorithms have similar security strength for curves with similar key lengths. The EdDSA algorithm is slightly faster than ECDSA for the most popular curves like edwards25519 and edwards448. Unlike ECDSA, it is not possible for EdDSA to recover the signer's public key from the signature and the message.

EdDSA is a digital signature system with 11 parameters. The generic EdDSA digital signature system with its 11 input parameters is not intended to be implemented directly. Choosing parameters is critical for secure and efficient operation. Instead, you would implement a partic- ular parameter choice for EdDSA (such as Ed25519 or Ed448). Therefore, a precise explanation of the generic EdDSA is thus not particularly useful for implementers. For background and completeness, a succinct description of the generic EdDSA algorithm is given on RFC8062.

Public Parameters

- Finite field $\mathbb{F}_q$ over odd prime power q.
- Elliptic curve E over $\mathbb{F}_q$ whose group $E(\mathbb{F}_q)$ of $\mathbb{F}_q$-rational points has order $\#E(\mathbb{F}_q) = 2^c\ell$ where ℓ is a large prime and 2^c is called the cofactor.
- of base point $G \in E(\mathbb{F}_q)$ with order ℓ.
- of has function H with $2b$-bit outputs, where $2^{b-1} \geq q$ so that elements of $\mathbb{F}_q$ and curve points in $E(\mathbb{F}_q)$ can be represented by strings of b bits.

Keys

- Secret key: b-bit random string x (where $b = 256$).
- $H(x) = \text{SHA512}(x) = (h_0, ..., h_{2b-1})$.
- Derive integer $s = 2^{b-2} + \sum_{3 \leq i \leq b-3} 2^i h_i$ (s is a multiple of 8).
  - Use $H_L(x)$ the first half of $H(x)$ to generate the public key by setting the first three bits of the first octet and the last bit of the last octet to zero and setting the second last bit of the last octet to one. Hence, we set $h_0 = h_1 = h_2 = h_{b1} = 0$ and $h_{b2} = 1$. Determine from this new bit string as an integer $s \in \mathbb{F}_q$ using little-endian convention.
- Compute $P = s.G$.
  - The public key is encoded as compressed EC point: the y-coordinate, combined with the lowest bit (the parity) of the x-coordinate. For Ed25519 the public key is 32 bytes. For Ed448 the public key is 57 bytes
- Public key: Encoding $\tilde{P}$ of $P = (x_P, y_P)$ as y_P and one (parity) bit of x_P (needs b bits).
- Compute P from $\tilde{P} : x_P = \pm\sqrt{(y_P^2 - 1)/(dy_P^2 + 1)}$.
  - The private key is generated from a random number, called seed (which should have similar bit length, like the curve order). The seed is first hashed, then the last few bits, corresponding to the curve cofactor (8 for Ed25519 and 4 for X448) are cleared, then the highest bit is cleared and the second highest bit is set. These transformations guarantee that the private key will always belong to the same subgroup of EC points on the curve and that the private keys will always have similar bit length (to protect from timing-based side-channel attacks). For Ed25519 the private key is 32 bytes. For Ed448 the private key is 57 bytes.

HashEdDSA Signing

- Input: (secret key x, message: M).
- Compute $(h_0, ..., h_{b-1}) = H(x)$.
- Derive integer $s = 2^{b-2} + \sum_{3 \leq i \leq b-3} 2^i h_i$ (s is a multiple of 8).
- $H_R(x) = h_b||h_{b+1}||...||h_{2b-1}$
- Deterministically compute $r = SHA512(H_R(x)||SHA512(M)) \in \{0, ..., 2^{2b} - 1\}$.
  - This r will be 64-octets long, and we treat it as a little-endian integer modulo q.
- Calculate the public key point behind r by multiplying it by the curve generator: $R = r.G$.
- Calculate $h = SHA512(\tilde{R}||\tilde{P}||SHA512(M))$.
- Calculate $S = r + hs \bmod \ell$.
- Return the signature $(\tilde{R}, \tilde{S})$, with $\tilde{S}$ the b-bit encoding of S.
  - The digital signature is 64 bytes (32 + 32 bytes) for Ed25519 and 114 bytes (57 + 57 bytes) for Ed448 (confirming that the signer knows m and x).

Verification

- Signature $(\tilde{R}, \tilde{S})$, public key $\tilde{P}$, message M.
- Parse P from $\tilde{P}$, R from $\tilde{R}$, and S from $\tilde{S}$.
- Verify that s lies in the half open interval [0, ℓ]
- Calculate $h = SHA512(\tilde{R}||\tilde{P}||SHA512(M))$.
- Calculate $2^3 S.G = 2^3.R + 2^3 h.P$.
- Check $S' \stackrel{?}{=} S$ in E (Rejects if parsing fails or equation does not hold).

3.4 Threshold EdDSA Signing Algorithm

3.4.1 Shamir Secret Sharing Scheme with a Dealer

In a (t, n) secret sharing scheme, a dealer distributes a secret s to n players $\mathcal{P}_1, ..., \mathcal{P}_n$ in such a way that any group of at least t players can reconstruct the secret s, while any group of less than t players do not get any information about s.

Secret sharing

- A dealer chooses $s \in_R \mathbb{Z}_q$(where $n < q$).
- The dealer chooses a random polynomial $f(.)$ over $\mathbb{Z}_q$ of degree at $t-1$ satisfying $f(0) = s$.
- Each player $\mathcal{P}_i$ receives $s_i = f(i)$ as his share.
- The dealer computes the public key of the players as $P = s.G$.
- The public and private key shares: $(pk, (sk_1, ..., sk_n)) = (P, (s_1, ..., s_n))$.

Secret Reconstruction An arbitrary group $\mathcal{P}$ of t participants can reconstruct the polynomial $f(.)$ by Lagrange's interpolation as follows:

$$f(u) = \sum_{i \in \mathcal{P}} f(i)\omega_i(u) \text{ where } \omega_i(u) = \prod_{j \in \mathcal{P}, j \neq i} \frac{u-j}{i-j} \bmod q. \tag{4}$$

Since it holds that $s = f(0)$, the group $\mathcal{P}$ can reconstruct the secret as follows:

$$s = f(0) = \sum_{i \in \mathcal{P}} f(i)\omega_i(u) \text{ where } \omega_i = \omega_i(0) = \prod_{j \in \mathcal{P}, j \neq i} \frac{j}{j-i} \bmod q. \tag{5}$$

3.4.2 Verifiable Secret Sharing Schemes

We use elliptic curve notation for the discrete logarithm problem. Suppose q is a large prime and G, H are generators of a subgroup of order q of an elliptic curve E. We assume that E is chosen in such a way that the discrete logarithm problem in the subgroup generated by G is hard, so it is infeasible to compute the integer d such that $G = dH$.

Verifiable Secret Sharing Scheme to Prevent the Dealer From Cheating A Verifiable Secret Sharing scheme (VSS) prevents the dealer from cheating. In a VSS scheme, each player can verify his share. If the dealer distributes inconsistent shares, he will be detected.

Assume the dealer has a secret $s' \in \mathbb{Z}_q$ and a random number $s' \in \mathbb{Z}_q$, and is committed to the pair (s, s') through public information $C_0 = sG + s'H$. The secret s can be shared among $P_1, ..., P_n$ as follows.

The dealer performs the following steps:

1. Choose random polynomials $f(u) = s + f_1 u + ... + f_{t-1}u^{t-1}$ and $f'(u) = s + f'_1 u + ... + f'_{t-1}u^{t-1}$ where $s, s', f_k, f'_k \in \mathbb{Z}_q$ for $k \in \{1, ..., t-1\}$
2. Compute $(s_i, s'_i) = (f(i), f'(i))$ for $i \in \{1, ..., n\}$
3. Send (s_i, s'_i) secretly to player $\mathcal{P}_i$ for $i \in \{1, ..., n\}$

4. Broadcast the values $C_k = f_k G + f'_k H$ for $k \in \{1, ..., t-1\}$

Each player $\mathcal{P}_i$ performs the following steps:

1. Verify
$$s_i G + s'_i H = \sum_{k=0}^{t-1} i^k C_k. \tag{6}$$
If this is false, broadcast a complaint against the dealer.

2. For each complaint from a player i, the dealer defends himself by broadcasting the values $f(i), f'(i)$ that satisfy the above equality in step 1. Reject the dealer if
   - he received at least t complaints in step 1, or
   - he answered to a complaint in step 2 with values that violate step 1.

   Pedersen proved that any coalition of less than t players cannot get any information about the shared secret, provided that the discrete logarithm problem in E is hard.

Generating a Random Secret without Trusted Dealer We can also generate a random shared secret in a distributed way which can be achieved by the following protocol. The *KeyGen* protocol will be represented as

$$(s_1, ..., s_n) \xleftarrow{(t,n)} KeyGen((r|Y, a_k G, H_0), k \in \{1, ..., t-1\}) \tag{7}$$

The notation here means that:

- H_0: the set of players that have not been detected to be cheating.
- s_j is $\mathcal{P}_j$'s share of the secret r for each $j \in H_0$.
- The values $a_k G$ are the public commitments of the sharing polynomial $f(\Delta)$ (they can be computed using public information).
- $(r, Y) : r$ is a private key and Y is the corresponding public key.

KeyGen():

1. Each player $\mathcal{P}_i$ chooses $r_i, r'_i \in \mathbb{Z}_l$ at random and verifiably secret shares (r_i, r'_i), acting as the dealer according to Pedersen's VSS scheme. Let the sharing polynomials be $f_i(u) = \sum_{k=0}^{t-1} a_{ik} u^k$ and $f'_i(u) = \sum_{k=0}^{t-1} a'_{ik} u^k$ where $a_{i0} = r_i, a'_{i0} = r'_i$, and let the public commitments be $C_{ik} = a_{ik} = a_{ik} G + a'_{ik} H$ for $k \in \{0, ..., t-1\}$.

2. The distributed secret value r is not explicitly computed by any player, but it equals $r = \sum_{i \in H_0} r_i$. Each player $\mathcal{P}_i$ sets his share of the secret as $s_i = \sum_{j \in H_0} f_j(i) \bmod q$, and the values $s'_i = \sum_{j \in H_0} f'_j(i) \bmod q$.

3. Extracting $Y = \sum_{j \in H_0} r_j G$: Each player in H_0 exposes $Y_i = s_i G$ via Feldman's scheme:

   - Each player $\mathcal{P}_i$ for $i \in H_0$ broadcasts $A_{ik} = a_{ik} G$ for $k \in \{0, ..., t-1\}$.
   - Each player $\mathcal{P}_i$ verifies the values broadcast by the other players in H_0. In particular, every $\mathcal{P}_i$ for $i \in H_0, \mathcal{P}_j$ checks if

     $$f_i(j)G = \sum_{k=0}^{t-1} j^k A_{ik}. \tag{8}$$

     If the check fails for an index $i, \mathcal{P}_j$ complains against $\mathcal{P}_i$ by broadcasting the values $f_i(j), f_i'(j)$ that satisfy (6) but do not satisfy (8).
   - For players $\mathcal{P}_i$ who received at least one valid complaint, i.e., values which satisfy (6) but do not satisfy (8), the other players run the reconstruction phase of Pedersen's VSS scheme to compute $r_i, f_i(.), A_{ik}$ for $k = 0, ..., t-1$ in the clear. All players in H_0 set $Y_i = r_i G$.

   After executing this protocol, the following equations hold:

   $Y = rG$
   f(u) = r + $a_1 u + ... + a_{t-1} u^{t-1}$ where $a_k = \sum_{j \in H_a} a_{jk}$ for $k \in \{1, ..., t-1\}$
   $f(j) = s_j$ for $j \in H_0$.

   This scheme has been proven to be robust under the assumption that $t \leq n/2$, i.e., if less than t players are corrupted, the values computed by the honest players satisfy the above equations.

3.4.3 Threshold EdDSA

Public Parameters

- Finite field $\mathbb{F}_q$ over odd prime power q.
- Elliptic curve E over $\mathbb{F}_q$ whose group $E(\mathbb{F}_q)$ of $\mathbb{F}_q$-rational points has order $\#E(\mathbb{F}_q) = 2^c \ell$ where ℓ is a large prime and 2^c is called the cofactor.
- of base point $G \in E(\mathbb{F}_q)$ with order ℓ.
- of has function H with $2b$-bit outputs, where $2^{b-1} \geq q$ so that elements of $\mathbb{F}_q$ and curve points in $E(\mathbb{F}_q)$ can be represented by strings of b bits.

Key Generation

- Execute the $KeyGen$ algorithm presented in Section 3.4.2 and compute $(\alpha_1, ..., \alpha_n) \xleftarrow{(t,n)} KeyGen((SK|PK), b_kG, H_0)k \in \{1, ..., t-1\})$.
- For each $j \in H_0, \alpha_j$ is the secret key share of $\mathcal{P}_j$ and will be used to issue a partial signature for the key pair (PK, SK) where $j \in H_0, k \in \{1, ..., t-1\}$.

HashEdDSA Signing

Suppose the players with index set $H_1 \subset H_0$ want to issue a signature on a message M

- If $|H_1| < t$, stop. Otherwise, H_1 generates a random shared secret as shown in Section 3.4.2. Let the public output be

$$(\beta_1, ..., \beta_n) \overset{(t,n)}{=} KeyGen(e|V, c_kG, H_2) \text{ where } k \in \{1, ..., t-1\}. \tag{9}$$

- If $|H_2| < t$, stop. Otherwise, each P_i for $i \in H_2$ reveals

$$\gamma_i = \beta_i + SHA512(V||PK||SHA512(M))\alpha_i \bmod \ell. \tag{10}$$

- Each P_i for $i \in H_2$ verifies that

$$\gamma_j G = V + \sum_{k=1}^{t-1} c_k j^k G + SHA512(V||PK||SHA512(M))(PK + \sum_{k=1}^{t-1} b_k j^k G) \text{ for all } j \in H_2. \tag{11}$$

  Let H_3 be the index set of players not detected to be cheating at step 3.

- If $|H_3| < t$, stop. Otherwise each P_i for $i \in H_3$ selects an arbitrary set $H_4 \subset H_3$ with $|H_4| = t$ and computes σ satisfying $\sigma = e + SHA512(V||PK||SHA512(M))SK$ by

$$\sigma = \sum_{j \in H_4} \gamma_j \omega_j \text{ where } \omega_j = \prod_{l \neq j l \in H_4} \frac{l}{l-j} \tag{12}$$

  The signature is (σ, V), which every player sends to the user. The signature can be verified as in Schnorr's original scheme:

$$\sigma G = V + SHA512(V||PK||SHA512(M))PK \text{ and } \sigma \in \mathbb{Z}_q. \tag{13}$$

Correctness

We have to show that the signature σ computed in Step 4 is the EdDSA signature on M, i.e.,

$$\sigma = e + SHA512(V||PK||SHA512(M))SK \tag{14}$$

- Let $F_1(\delta)$ be the sharing polynomial of the key generation protocol ($\alpha_i = F_1(i)$ for $i \in H_0$), and let $F_2(\delta)$ be the sharing polynomial implied by the generated random shared secret in Step 1 ($\beta_i = F_2(i)$ for $i \in H_1$).
- Furthermore, let $F_3(\delta) := F_2(\delta) + SHA512(V||PK||SHA512(M))F_1(\delta)$. Since $\gamma_i = F_3(i)$ for $i \in H_3$, it follows from Lagrange's interpolation formula that the players compute $\sigma = F_3(0)$.

We can now argue as follows:

$$\begin{aligned}\sigma &= F_3(0) \\ &= F_2(0) + SHA512(V||PK||SHA512(M))F_1(0) \\ &= e + SHA512(V||PK||SHA512(M))SK.\end{aligned}$$

We have to show that if less than t players are corrupted, the scheme always produces a valid signature. We assume that $t \leq n/2$.

3.5 Secure Multi-Party Computation (SMPC)

In secure multi-party computation, there are n parties, where each of them holds a private input x_i. They wish to jointly and securely perform a computation based on their inputs that maps pairs of inputs (one input for each party) to pairs of outputs (one output for each party) [7, 2]. Such a process is called as the desired functionality, and denoted

$$f : \{0,1\}^* \times ... \times \{0,1\}^* \to \{0,1\}^* \times ... \times \{0,1\}^* \tag{15}$$

where $f = (f_1, ..., f_n)$. Note that the term functionality is used rather than function, since the parties of a protocol have separate inputs, and, in general, wish to compute different functions of the common input. At the end of the computation, both parties should know the output of f. That is, for every pair of inputs $(x_1, ..., x_n)$, the desired output-pair for the i-th party is $f_i(x_1, ..., x_n)$ ranging over pairs of strings.

A protocol is a secure multi-party protocol if the following security properties hold:

- **Correctness:** The output that the parties receive is guaranteed to be correct. In other words, no party can influence the output of the computation other than by changing its own input.
- **Privacy:** No party should learn anything more than what is implied by the function f and

its own input. In other words, no party can obtain information about the honest party's input and what it sees from the protocol execution except what it can derive from its own input and the output.

- **Fairness:** A corrupted party should receive its output if and only if the honest party also receives its output. In general, if a party is corrupted, then the adversary may learn the output before the honest party obtains it and then may decide to abort the protocol. This may result in an unfair situation.

3.6 BIP32 - Hierarchical Deterministic Wallets

BIP32 is a mechanism for deriving a tree of key-pairs from a single cryptographically strong seed. Namely, it is called hierarchical due to its tree-like structure which can be used to express additional organizational meaning. All the child keys are derived from a single master key which is known as the seed. Trivially, the keys are deterministically related to each other and can be recovered again if one has the original seed. Therefore, this structure prevents outdated backups. One of the most important advantage is that crypto-wallets can be shared partially or entirely with different (sub)systems, each with or without the ability to spend coins. Given a parent extended key and an index i, it is possible to compute the corresponding child extended key as follows:

3.6.1 Private parent key → private child key

A child extended private key can be generated from a parent extended private key as follows:

1. $CKDpriv((K_{par}, cpar), i) \rightarrow (k_i, c_i)$.
2. Computes a child $xpriv$ from the parent $xpriv$.
3. Check whether $2^{32} - 1 \geq i \geq 2^{31}$ (if the child is a hardened key).
   - If so, HMAC-SHA512$(chaincode, k, i) \rightarrow I$.
   - If not, $I =$ HMAC-SHA512$(chaincode, point(k), i) \rightarrow I$.
4. Split I into two 32-byte sequences, I_L and I_R.
5. The returned child key $k_i = I_L + K_{par}$ mod n.
6. The returned chain code c_i is I_R.

3.6.2 Public parent key → public child key

A child public key can be generated from a parent public key as follows:

1. $CKDpub((K_{par}, cpar), i) \rightarrow (K_i, c_i)$.
2. Check whether $i \geq 2^{31}$ (if the child is a hardened key).

- If so (hardened child), return failure.
- If not (normal child), $I = \text{HMAC-SHA512}(chaincode, point(k), i) \rightarrow I$.

3. Split I into two 32-byte sequences, I_L and I_R.
4. The returned child key $K_i = I_L + K_{par}$ mod n.
5. The returned chain code c_i is I_R.

4 Entities of Polka.Domain

- **Signatories:** They are responsible for generating a signature for users' domains. They are assumed to be trusted organization for a better trust distribution among multiple entities. Polka.Domain aims to decentralize the trust according to the standards should enforce the selection of members from politically and geographically disparate entities (e.g., certificate authorities, browsers, reputable companies/foundations, research institutes, universities).
- **Users:** They are the owner their domains and crypto wallets.
- **Miners:** They are part of the underlying blockchain and are writers which select pending transactions from the pool, validate them, and then create new blocks according to the underlying consensus protocol. They generate blocks for all the transactions of the underlying blockchain and propagate them to the P2P network.
- **Blockchains:** They are different blockchains which have different cryptocurrencies. These cryptocurrency addresses will be added to the domains of users.

5 Asset Management Functionalities of Polka.Domain

In this section, we will describe main functionalities of Polka.Domain. Polka.Domain will not only provide reliability through decentralized trusted certification but also privacy of users' cryptocurrency wallets.

5.1 Generating a Certificate for Polka.Domain Authority Council

1. A certificate for a set of authority council will be generated using (k, n)-dynamic threshold signature scheme (where private key is not changed in case of removing members or adding new members) to form a Distributed Certificate Authority, what we call Polka.Domain Authority Council (DAC). This council is set up by choosing n parties in the network. The private key SK of the DAC is thus shared among DAC.

   (a) Each party of DAC P_i has a unique identifier (ID) and is able to communicate with other parties of the council.

(b) Each i-th party holds a private key share SK_i corresponding to its public key pk_{P_i}.

(c) At the end of the threshold key generation ceremony, CAD will compute their common public key pk_{DAC} which is self-signed by the participants (note that no one knows the corresponding private key sk_{DAC}).

2. pk_{DAC} with its root certificate will be added to the chain as a transaction.

3. pk_{DAC} is used to add new domains to the chain and revoke them.

5.2 Registration to Polka.Domain

1. A user opens the Polka.Domain wallet.

2. The user enters his/her phone number. The wallet sends a random number as an SMS.

3. The user enters this number to the wallet.

4. The wallet shows a backup recovery phrase page which shows a list of mnemonic words. The user is supposed to save this mnemonic for backup purposes. The user clicks Next to continue.

5. A new page is shown to verify recovery phase. The user chooses the mnemonic words from the list correctly in order, and clicks Next to continue.

6. A new page asks the user to set a password to secure these mnemonic words. Namely, the password will be used as an extra layer of security when the user interacts with his/her wallet.

7. A Polka.Domain address will be created for the user.

5.3 Encryption of the Signing Key

In the next protocol, we aim to protect the partial private key share $SignKey$ which will be used to partially sign the transactions.

1. Install Polka.DomainMobile to your mobile device.

2. Login to Polka.DomainMobile

3. Polka.DomainMobile generates a 256 bit key K_{mobile} and stores locally.

4. Login to Polka.DomainDesktopWeb.

5. Generate an ephemeral (short-term) private/public key pair (pk,sk) and generate QR code of the public key (using EC-ElGamal encryption scheme).

6. Polka.DomainMobile reads the QR code and obtains the public key.

7. The signer enters the password (pw) into Polka.DomainMobile app. Then, the app computes the trapdoor ($K_{trap} = argon2(pw, K_{mobile})$). Finally, Polka.DomainMobile computes $C = Enc_{pk}(K_{trap})$ and sends it to the PariCold server.

8. The PariCold server forwards this ciphertext to Polka.DomainDesktopWeb.

9. Polka.DomainDesktopWeb first computes $K_{trap} = Dec_{sk}(C)$ and computes the symmetric key ($K = argon2(pw, K_{trap})$)

10. Finally, Polka.DomainDesktopWeb encrypts the signing key $SignKey$ as $Enc_K(SignKey)$

11. Polka.DomainDesktopWeb deletes all the information except $Enc_K(SignKey)$.

5.4 Obtaining the Signing Key

1. Login to Polka.DomainDesktopWeb. The signing key $SignKey$ is locally stored as $C = Enc_K(SignKey)$ where K is a symmetric key.

2. Generate an ephemeral (short-term) private/public key pair (EC-ElGamal) and generate QR code of the public key.

3. Polka.DomainMobile reads the QR code and obtains the public key.

4. The signer enters the password (pw) into Polka.DomainMobile. Then, Polka.DomainMobile com- putes the trapdoor ($K_{trap} = argon2(pw, K_{mobile})$) where K_{mobile} was already stored in Polka.DomainMobile.

5. Finally, Polka.DomainMobile encrypts K_{trap} with the received public key and sends it to the PariCold server.

6. The PariCold server sends this ciphertext to Polka.DomainDesktopWeb.

7. Polka.DomainDesktopWeb recovers the trapdoor key K_{trap} and computes the symmetric key ($K = argon2(pw, K_{trap})$)

8. Finally, Polka.DomainDesktopWeb decrypts $Enc_K(SignKey)$ and obtains the signing key $SignKey$.

5.5 Creating a Domain on Polka.Domain

A Polka.Domain domain will be created as follows:

1. A user U generates a fresh public and private key pair (pk_U, sk_U).

2. User U keeps sk_U private.

3. User creates a Certificate Signing Request (CSR). The CSR contains information identifying the user U (such as a distinguished name in the case of an X.509 certificate) which must be signed using U's private key sk_U . The CSR also contains the public key of U.

4. The user encrypts his/her credentials with the public key of Polka.Domain Authority Council (i.e., $Ciphertext = Enc_{pk_{DAC}}(Credentials_U)$ where $Credentials_U$ contains a name & surname, date of birth, address, mobile number, email).

5. User U submits his/her public key together with a domain (e.g., mydomain.pd) to Polka.Domain to be certified. Namely, $(CSR, domain, Ciphertext)$ is submitted as a new transaction to Polka.Domain.

6. Polka.Domain Authority Council first validates the signature using its public key pk_U.

7. Polka.Domain Authority Council checks whether the domain exists

   - domain extension is .pd
   - on Polka.Domain chain
   - in the reserved list of the Alexa top 100K domains (for legacy DNS owners).
   - on existing chains (e.g., .zil on Zilliqa, .crypto on Ethereum).

8. If the checks pass then the Polka.Domain Authority Council executes a threshold signing ceremony. At the end of the ceremony, a signature

$$Signature_{Domain,U} = Sign_{DAC}(..., pk_U, domain, ...) \tag{16}$$

   will be generated.

9. The signed domain $Signature_{Domain,U}$ is submitted to the chain as a transaction.

10. A smart contract of Polka.Domain on the chain (which contains the public key of Polka.Domain) checks the signature and confirms the transaction.

5.6 Renewing/Revoking Name Ownership of a Domain on Polka.Domain

- If membership is not renewed before the renewal date, it will be automatically expired on the expiration date.
- At any point before the renewal date, the owner can revoke his domain with his signing key.
- If the owner loses his signing key, he can ask Polka.Domain Authority Council to revoke the user's domain certificate before the renewal date.
- In case of revocation, the domain will be reserved to the previous owner for three days.
- After the grace period, revoked domains become available for public registration.

5.7 Transferring Name Ownership of a Domain on Polka.Domain

A user may transfer a domain from the connected primary address to bind with another address through Polka.Domain. This transaction can be authorized by either a domain owner, an operator, or an approved address. Defines by the ERC-721 standard, a direct owner of a domain has full control in managing domain ownership and records. Operators can control all domains owned by a user and there can be multiple operators per user. A domain owner can set an approved address that can control one particular domain.

5.8 Top-level Domain Management

A top-level domain is at the highest level in the hierarchical DNS after the root domain (eg, .com, .eth, .pd). The governance module will vote on whether to make a top-level domain available for registration, and the users can register as a top-level domain's second-level domain.

5.9 Revocation of a Cryptocurrency Address from a Domain

Only the user (with his signing key) can revoke the user's cryptocurrency wallet address. Once a domain is revoked all cryptocurrency addresses will not be reliable anymore.

5.10 Adding a Cryptocurrency Address to an Existing Domain on Polka.Domain with No Privacy

A user may have more than one crypto wallet addresses. These addresses can be added/integrated to an existing domain as follows.

1. A user U generates a public and private key pair for a cryptocurrency address

$$(pk_{wallet_i,U}, sk_{wallet_i}, U) \tag{17}$$

   according to BIP32 standards where $pk_{wallet_i,U}$ is a public key address of i-th wallet address of U.

2. The user U computes

$$Signature_{wallet_i,U,Domain} = Sign_s k_U(pk_{wallet,U,Domain}) \tag{18}$$

   where sk_U is his signing key of his domain $Domain$.

3. The user U submits the signature $Signature_{wallet_i,U,Domain}$ as a transaction to Polka.Domain.

4. Anyone who knows the domain of the user can learn and obtain his $pk_{wallet,U}$ without having to communicate with the user.

5.11 Adding a Cryptocurrency Address to an Existing Domain on Polka.Domain with Privacy

1. A user U generates a parent public and private key pair (according to BIP32 standards) for a cryptocurrency address and wants his parent public key together with his domain to Polka.Domain to be certified in a privacy manner. The user signs this request and submits to Polka.Domain.

2. Polka.Domain Authority Council first checks whether the domain exists on the chain, and if exists, he obtains the user's domain public key and then validates the signature as well as the public key of the crypto wallet.

3. If the checks pass then the Polka.Domain Authority Council executes a SMPC ceremony.

4. At the end of SMPC, the Polka.Domain Authority Council altogether computes encrypted and signed parent key which is concatenated with the domain (i.e., obfuscated parent key and his domain).

5. In particular,the Polka.Domain Authority Council generates

$$Sign_{sk_{DAC}}(Enc_{pk}(xpubkey), mydomain.pd) \tag{19}$$

   where sk_{DAC} is the private key of the Authority Council, $xpubkey$ is the parent public key of the user, and $user.Polka.Domain$ is the domain of the user.

6. The encrypted and signed message is submitted to the Polka.Domain chain as a transaction.

7. Everyone can validate the signature that it has been indeed signed by a valid and decentralized authority as well as can see the user's domain address. However, no one will be able to learn any information about the parent key.

5.12 Requesting a Child Key of a Cryptocurrency for a Domain

A sender can request a child key of a cryptocurrency which belongs to a domain (i.e., a recipient). Polka.Domain executes SMPC to generate a signed child key and send it back to the user. This provides at least three major advantages:

- Senders will only need to know the domain of a recipient. They do not need to know and use cryptocurrency addresses.
- In case of sender is corrupted: The sender cannot trace recipient's transactions.
- No one knows the parent key of recipients whose privacy is preserved.

5.12.1 Funds are always Safe with Polka.Domain!

If a cryptocurrency address is not listed under a domain the transaction will be rejected. Therefore, the funds will always be safe and transfer of funds to an incorrect address is now completely over.

6 Advanced Features

In addition to domain and asset management functionalities, Polka.Domain strive to provide solutions to the problems of conventional domain name and make blockchain domains commonly usable and accessible as we move towards the decentralized Web 3.0.

6.1 Domain Marketplace

Decentralized domains are represented as tokenized assets in the form of Non-Fungible Tokens. They are unique, not interchangeable, collectible, and tradable. Newly registered domain becomes a NFT and stored as an asset automatically in your wallet. The NFT metadata is stored on the IPFS distributed file system. By building a decentralized one-step domain exchange and auction platform, users can list and sell their domains at a fixed price (fixed swap) or through an auction in the marketplace. After a domain has been transferred/sold, the new owner has the full ownership of the NFT.

6.2 Cross-chain transaction

Users can start an on-chain transaction through their domain. As one initiate transactions using Polka.Domain, the domain management module looks for the corresponding sending and receiving addresses, generate transaction data, and broadcast transaction. If the receiving the address is on a different blockchain network, the system generates cross-chain data through Polkadot XCMP. Unlike Bitcoin and Ethereum, Polkadot enables any data to be sent between any blockchain, in turn allowing for multiple transactions to be processed in parallel under shared security with reasonable transaction fees. Driving mainstream adoption while upholding the blockchain's security standards.

6.3 Censorship Resistant Websites

Around the world, resources are being increasingly censored or taken down from the internet. Centralized data repositories can monitor your browsing activities and have access to your personal data. Unsurprisingly these entities are also prone to hacks and attacks. Due to the nature of decentralization, none of the data is owned by one centralized entity; there's no authority with the power to enforce censorship on any blockchain domain names. The domains are stored in your wallet, protected by their private key. Thus, only you, the owner, can take down or deny access to your domain. Furthermore, you can securely process payments without involving traditional, third-party payment gateways.

7 Related Work

There already exist some decentralized, permissionless naming protocols where every peer is validating and in charge of managing the root DNS naming zone to create an alternative to existing Certificate Authorities and naming systems. We briefly describe security issues associated with some of these projects and how we wish to build on and improve with Polka.Domain.

Some blockchain-based DNS projects involve no identity verification and no adequate solution if both online and offline keys are compromised. For projects on Bitcoin, verifying the owner of a public key, and looking up the public key of an identity is extremely inefficient. Some projects are not cryptocurrency agnostic and do not support domain extensions on other chains. Others involve building a registry of domains that empowers individuals to register for specific blockchain domains, but they are not privacy-preserving. And more recently, the high Ethereum gas fee has made it impossible for ERC-20 projects to run any microtransaction payments on Ethereum. Polka.Domain aims to secure decentralized domains through decentralized certificate transparency based on the Polkadot network while preserving end-users privacy.