OOPS: One-time Oblivious Polynomial Signatures

Kobi Gurkan
kgurkan@baincapital.com

Philipp Jovanovic
p.jovanovic@ucl.ac.uk

Andrija Novakovic
anovakovic@baincapital.com

December 2025

Abstract

We introduce *one-time oblivious polynomial signatures (OOPS)*, a signature scheme based on polynomials over pairing-based elliptic curves that can securely produce signatures for up to a threshold of n different messages. Signing more than n messages allows anyone to forge signatures under the given parameters, making it necessary to reparameterize the scheme occasionally. We show that this property is not a severe limitation though by demonstrating how to build various efficient OOPS-based cryptographic protocols, including delegatable signatures, 1-out-of-n oblivious transfer, and partially oblivious PRFs.

1 Introduction

One-time oblivious polynomial signatures allow signers to securely sign up to a threshold of n messages for a given set of parameters that signer and verifiers have agreed upon ahead of time, e.g., during a setup phase. If the signer issues more than n signatures, anyone can forge signatures on new messages under the given parameters (without knowing the signers secret key). To keep producing signatures securely, a signer thus needs to refresh the parameters of the signature scheme after signing up to n messages. This property arises from the fact that the OOPS scheme is based on the evaluation of polynomials (over pairing-based elliptic curves) of degree n.

Even though this property might seem limiting at first, there are various situations where it is actually quite natural or even desireable to limit the number of signatures that a signer can or should produce. For example, in the context of identity systems [Ros+23, Whi19]. Concrete examples include systems like Privacy Pass [Int25], for anonymous authentication, or proof of stake blockchains where equivocation by validators results in slashing [GHW23]. As we will show later, OOPS can be used as a building block for constructing efficient cryptographic protocols including delegatable signatures, oblivious transfer, and (partially) oblivious PRFs.

Our construction is conceptually simple. The signer and the verifier agree, for a given context/topic (often identified through a seed s), on a set of elliptic curve base points $H_0, ..., H_n$, whose discrete logarithm relative to the group generator and each other is unknown. We treat

these base points as coefficients of a polynomial $H(X)$ and the signer produces a signature by evaluating the polynomial at the message m as $H(m) = \sum_{i=0}^{n} m^i \cdot H_i$ and multiplying by their secret key κ to obtain the signature $\sigma = \kappa H$. Intuitively, this signature scheme has unforgeability because knowing n evaluations on an n-degree polynomial does not allow you to evaluate the polynomial at another point.

Note that the construction can be extended directly to vectors of messages.

2 Related work

The construction is inspired by signatures for linear subspaces [Bon+09], where signers wish to produce signatures on base vectors of a linear subspace, such that the signature is valid for any vector in that space. In our construction, we wish to sign specific messages and vectors, while limiting the amount of times the signing key can be used for each epoch.

Semaphore RLN [Whi19] and zk-creds [Ros+23] use a special kind of *nullifiers*, where reusing the same key for an epoch, or a topic, results in the private key being revealed. Our construction achieves similar results in terms of rate-limiting at the cost of sacrificing unlinkability. We leave for future work how to achieve cloning resistance without using SNARKs for every message inside an epoch.

BLS multisignatures [BDN18] are constructed similarly, with the notable difference that they are unforgeable for any given number of messages, mainly due to the fact that each *message* is hashed to the group, rather than the slot description.

Signature delegation by Schnorr signatures is done in [AN24]. Constructions are similar in the amount of information that has to be shared between delegator and delegatee. A major difference in our approach is that the delegation is invisible to the verifier and looks the same as any other signature, where as in [AN24], while the proxy signer's identities are hidden, multiple Schnorr signatures are verified.

Oblivious Pseudorandom Functions (OPRFs) are a cryptographic primitive that allows a client to compute the output of a pseudorandom function (PRF) on a given input without revealing the input to the server that holds the PRF key [Jar+17, JKR19]. In our OPRF approach, the server gives the client the information needed to evaluate the OPRF locally for a specific topic. Additionally, the client is restricted to evaluate the PRF a limited amount of times before the evaluation capability becomes available for anyone seeing enough OPRF evaluations.

Oblivious transfer protocols such as [CO15], [Kol+16] In our scheme, the protocol can be split into offline and online phase where most of the sender's work can be done during the offline phase. During the online phase both sender and verifier need to send only a constant amount of messages to each other.

BBS [BBS04, TZ23] and BBS+ [ASM08, CDL16] are designed for long-lived credentials and remain secure under an unbounded number of uses of the signing key. OOPS instead treats key reuse as a limited resource and enforces bounded usage per topic directly at the signature level, which in turn allowing it to use a non-randomized signing algorithm. BBS

signatures have a structural similarity to OOPS when considering a vector variant of OOPS for degree 1 polynomials, with a major difference of how the private key is used during signing.

3 Preliminaries

We work over pairing-friendly elliptic curves. Thus, we denote by E a pairing-friendly elliptic curve with scalar field $\mathbb{F}_r$ and groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ with generators $G_1 \in \mathbb{G}_1$ and $G_2 \in \mathbb{G}_2$ and that are equipped with an efficient bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$. We often use the notation $<,>$ to denote the evaluation of the pairing operation, *i.e.* $< P, Q >= e(P, Q)$ for $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$. In particular, we will work with type-3 pairings and we refer the reader to the literature for more details. Furthermore, let $\mathcal{H} : \{0,1\}^* \to \mathbb{G}_1$ denote a hash function that maps binary strings to points in $\mathbb{G}_1$. We also often write $X = [x]_i$ to denote the scalar multiplication of generator $G \in \mathbb{G}_i$ by a scalar $x \in \mathbb{F}_r$ for $i \in \{1, 2\}$.

We work in the Algebraic Group Model (AGM). We say that the algorithm is *algebraic* if its output can be expressed as a linear combination of the group elements that it has seen during its execution. The AGM assumes that the adversary is algebraic and that the only way to break the scheme is to produce an algebraic forgery. We assume that we work in a type-3 pairing setting, where $<,>: \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ and $\mathbb{G}_1 \neq \mathbb{G}_2$, such that when the adversary outputs an element $G \in \mathbb{G}_i$, the linear combination is over the elements they've seen in $\mathbb{G}_i$.

Our constructions rely on the (1,1)-Discrete Logarithm problem [BFL20]: Given a tuple $(G_1, \alpha G_1, G_2, \alpha G_2) \in \mathbb{G}_1^2 \times \mathbb{G}_2^2$ for $\alpha \in \mathbb{F}_r^*$, output α. We say that the (1,1)-DLog assumption holds in the AGM if no polynomial-time algebraic adversary can solve the (1,1)-DLog problem with non-negligible probability.

4 Definitions

4.1 One-time oblivious polynomial signature scheme

- KeyGen() $\to$ (sk, vk): Generates a secret key sk, and returns (sk, vk).
- GenerateTopic(d, topic) $\to T$: Takes as input a degree $d > 0$ and a topic description topic $\in \{0,1\}^*$ and generates and returns topic parameters T used for signing and verification.
- Sign(sk, m, d, topic) $\to \sigma$: Takes as input a secret key sk, a message m, and a topic. It generates topic parameters $T =$ GenerateTopic(d, topic) and uses them to return a signature σ.
- Verify(vk, m, σ, d, topic) $\to \{0,1\}$: Takes as input a verification key vk, message m, signature σ, d and topic. It generates topics parameters $T =$ GenerateTopic(d, topic) and checks whether σ is a valid signature. If it is, it returns 1 (valid), otherwise it returns 0 (invalid).

4.2 Delegatable signature scheme

- KeyGen() $\to$ (sk, vk): Generates a secret key sk, and returns (sk, vk).
- GenerateTopic(d, topic) $\to T$: Takes as input a degree $d > 0$ and a topic description topic $\in \{0,1\}^*$ and generates topic parameters T used for signing and verification.
- Delegate(sk, d, topic) $\to D$: Takes as input a secret key sk, a degree $d > 0$, and a topic description topic $\in \{0,1\}^*$. It generates delegations parameters D.
- Sign(D, m) $\to \sigma$: Takes as input delegation parameters D, and a message m. It uses D to return a signature σ.
- Verify(vk, m, σ, d, topic) $\to \{0,1\}$: Takes as input a verification key vk, message m, signature σ, d and topic. It generates topics parameters $T =$ GenerateTopic(d, topic) and checks whether σ is a valid signature. If it is, it returns 1 (valid), otherwise it returns 0 (invalid).

5 OOPS

We describe OOPS, a signature scheme that has inherent rate-limiting per topic. Signatures certify both a topic and a message under that topic. After revealing one too many signatures, an observer is able to forge signatures for any message under the same topic. We observe that this is a natural constraint occasionally. In proof of stake protocols, validators are prevented from signing multiple messages for the same block, which is enforced by a slashing, an economic constraint.

The OOPS signature scheme is then specified as follows:

- OOPS.Setup(1^λ) $\to (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, \mathbb{F}_r, G_1, G_2, \mathcal{H})$: Takes as input a security parameter λ and generates the parameters of the pairing pairing-friendly elliptic curve groups along with their generators and $\mathcal{H}$, the hash function to the group $\mathbb{G}_1$.
- OOPS.KeyGen() $\to$ (sk, vk): Samples a fresh secret key sk $= \kappa \in \mathbb{F}_r$, computes a verification key vk $= \kappa G_2$, and returns (sk, vk).
- OOPS.GenerateTopic(d, topic) $\to ((H_i)_{i=0}^d)$: Takes as input a degree $d > 0$ and a topic description topic $\in \{0,1\}^*$ and generates a list of base points $H_i = (H_0, ..., H_d)$ in $\mathbb{G}_1$ where $H_i = \mathcal{H}(\text{topic} \parallel d \parallel i)$ for $i \in \{0, ..., d\}$.
- OOPS.Sign(sk, m, d, topic) $\to \sigma$: Takes as input a secret key sk, a message m, and a topic. It derives vector of base points $\{H_i\}_{i=0}^d =$ OOPS.GenerateTopic(d, topic) and returns a signature $\sigma = \kappa(\sum_{i=0}^{n-1} m^i H_i) \in \mathbb{G}_1$.
- OOPS.Verify(vk, m, σ, d, topic) $\to \{0,1\}$: Takes as input a verification key vk, message m, signature σ, d and topic. It derives vector of base points $\{H_i\}_{i=0}^d =$ OOPS.

GenerateTopic(d, topic) and checks whether the following equation holds: $\langle \sigma, G_2 \rangle = \langle \sum_{i=0}^{n-1} m^i H_i, \ \text{vk} \rangle$. If it holds, it returns 1 (valid), otherwise it returns 0 (invalid).

5.1 Security proof

5.1.1 Existential unforgeability

Setup. The challenger runs OOPS.KeyGen gives the adversary $\text{vk} = \alpha G_2$.

Signature Queries. Adversary issues q_S signature queries to the challenger. Each query consists of a message, a topic and a degree. The challenger runs OOPS.Sign and responds with the signature. The adversary is allowed to call $\mathcal{H}$ for q_H times, which is also called inside OOPS. GenerateTopic.

Output. The adversary outputs a signature σ^* satisfying the verification equation for a tuple $(m^*, d^*, \text{topic}^*)$ such that the message m^* was not queried for the topic topic^* and degree d^* during the signature queries and additionally the number of signature queries for the topic topic^* and degree d^* is at most d^*.

5.1.2 Proof

The challenger is given a $(1,1) - \text{dlog}$ instance $(G_1, \alpha G_1, G_2, \alpha G_2) \in \mathbb{G}_1^2 \times \mathbb{G}_2^2$, and needs to output α. The challenger simulates the hashing and signatures oracles as following. For hash queries, the challenger samples $\rho_i \leftarrow \mathbb{F}_r$ and sets $\eta_i = \rho_i G_1$. For the specific hash queries related to topic bases, we set the special notation of: for query i on topic t_i and degree d_i, the challenger has $\rho_{t_i,d_i,j}$ for each $j \in [0, \deg]$ and has $\eta_i = H(t_i \parallel d_i \parallel j) = \rho_{t_i,d_i,j} G_1$. For signature query i with message m, topic t and degree d, the challenger responds with $\sigma_i = \sum_{j=0}^{d} m^j \rho_{t,d,j} \alpha G_1$. Eventually, the adversary outputs σ^* for a message m^* and topic t^* with a degree d^*. If the tuple satisfies the signature verification equation and if m^* was not queried more than d^* times for the topic t^* and degree d^*, the challenger can extract α as following.

Since the adversary is algebraic, we have:

$$\sigma^* = \gamma_{G_1} G_1 + \sum_{i=0}^{q_H - 1} \gamma_{H,i} \eta_i + \sum_{i=0}^{q_S - 1} \gamma_{\sigma,i} \sigma_i$$

$$= \gamma_{G_1} G_1 + \sum_{i=0}^{q_H - 1} \gamma_{H,i} \rho_i G_1 + \sum_{i=0}^{q_S - 1} \gamma_{\sigma,i} \sum_{j=0}^{d_i} m^j \rho_{t_i,d_i,j} \alpha G_1$$

Additionally given that the verification equation is satisfied we have:

$$\langle \sigma^*, G_2 \rangle = \left\langle \sum_{j=0}^{d^*} (m^*)^j H(t^* \parallel d^* \parallel j), \ \text{vk} \right\rangle$$

The challenger uses the two equations to extract α:

$$\alpha \sum_{j=0}^{d^*} (m^*)^j \rho_{t^*,\deg^*,j} = \gamma_{G_1} + \sum_{i=0}^{q_H-1} \gamma_{H,i}\rho_i + \sum_{i=0}^{q_S-1} \gamma_{\sigma,i} \sum_{j=0}^{d_i} m^j \rho_{t_i,d_i,j}\alpha$$

$$\alpha \left(\sum_{j=0}^{d^*} (m^*)^j \rho_{t^*,\deg^*,j} - \sum_{i=0}^{q_S-1} \gamma_{\sigma,i} \sum_{j=0}^{d_i} m^j \rho_{t_i,d_i,j} \right) = \gamma_{G_1} + \sum_{i=0}^{q_H-1} \gamma_{H,i}\rho_i$$

We argue that $S = (\sum_{j=0}^{d^*}(m^*)^j \rho_{t^*,\deg^*,j} - \sum_{i=0}^{q_S-1} \gamma_{\sigma,i} \sum_{j=0}^{d_i} m_i^j \rho_{t_i,d_i,j}) = 0$ with negligible probability. Because the adversary can query signatures with topic* and d^* up to d^* times, assuming that they did query d^* times, we can rewrite S as follows:

$$S = \sum_{j=0}^{d^*} (m^*)^j \rho_{t^*,\deg^*,j} - \sum_{i=0}^{d^*-1} \gamma_{\sigma,k_i} \sum_{j=0}^{d_{k_i}} m_{k_i}^j \rho_{t^*,d^*,j} + \sum_{i=0}^{q_S-1-d^*} \gamma_{\sigma,i} \sum_{j=0}^{d_i} m^j \rho_{t_i,d_i,j}$$

Observe that each signature corresponds to the evaluation of a polynomial, with coefficients $[\alpha\rho_{t_i,\deg_i,0}, ..., \alpha\rho_{t_i,\deg_i,\deg_i}]$ of degree d_i at a point m_i. Since the adversary queried at most d^* signatures for the topic t^* and degree d^* it has access to strictly less evaluations than the degree of polynomial defined by the base points $\{H(t^* \parallel d^* \parallel j)\}_{j=0}^{d^*}$. By intractability of polynomial interpolation, the adversary cannot evaluate the d^* polynomial at a new point from less than d^*+1 evaluations, thus the adversary can access another evaluation of the polynomial only if there is a linear dependency between the polynomial $[\alpha\rho_{t_i,\deg_i,0}, ..., \alpha\rho_{t_i,\deg_i,\deg_i}]$ and the polynomials defined by the signatures queried with different topics and degrees. However, since the coefficients of each polynomial are sampled uniformly at random from $\mathbb{F}_r$, the probability that such a linear dependency exists is negligible in the security parameter λ. Thus, with all but negligible probability $D = \sum_{j=0}^{d^*}(m^*)^j \rho_{t^*,\deg^*,j} - \sum_{i=0}^{d^*-1} \gamma_{\sigma,k_i} \sum_{j=0}^{d_{k_i}} m_{k_i}^j \rho_{t^*,d^*,j} \neq 0$ and the challenger can compute α as: $\alpha = (\gamma_{G_1} + \sum_{i=0}^{q_H-1} \gamma_{H,i}\rho_i) * D^{-1}$.

6 Delegatable signatures

In this section we show how one party can delegate signing rights, for a limited topic, to another party in a secure manner using our scheme. Consider two parties, a delegator and a delegatee. The delegator holds a private key κ and the corresponding verification key $\mathrm{vk} = [\kappa^{-1}]_2$. The delegatee wants to obtain signing rights from the delegator such that it can sign messages on behalf of the delegator without learning the private key κ. Additionally, we want to ensure a *topic*-limited delegation. In other words, the delegatee should only be able to sign a message in a specific context or a specific purpose. For example, the delegatee should only be able to sign messages for a specific time period or for a specific application. To achieve this, we identify each topic with an n degree polynomial $p(T) \in \mathbb{G}_1[T]$, such that the coefficients depend on the private key. A signature on a message m in the context of a topic is defined to be $p(m)$. The coefficients of the polynomial are known only to the delegator, thus nobody is able to evaluate the polynomial on their own. In order to delegate signing rights for

a specific topic, the delegator needs to send $n+1$ evaluations of the polynomial to the delegatee. More specifically, for a topic defined by a polynomial $p(T)$ of degree n, the delegator picks $n+1$ random points $t_0, ..., t_n \in \mathbb{F}_r^*$ and computes the evaluations $p(t_0), ..., p(t_n)$. The delegator then sends these evaluations to the delegatee. The delegatee can now interpolate the coefficients using these evaluations and can sign any message m by computing $p(m)$. By existential unforgeability of our scheme, the delegatee cannot derive the private key κ from these evaluations and therefore they can only sign messages in the context of the delegated topic. Note that most of the work of delegation can be done ahead of time, before knowing the specific delegatee, by publishing n evaluations. The delegator then only needs to send a single evaluation to the delegatee to complete the delegation.

6.1 Protocol

The construction is almost identical to OOPS, where the difference is in introducing a Delegate method and signing uses the delegation paramaters rather than the secret key.

- DOOPS.Setup, DOOPS.GenerateTopic, DOOPS.KeyGen and DOOPS.Verify are identical to the ones in OOPS.
- DOOPS.Delegate$(x, d, \text{topic}) \rightarrow ((E_i)_{i=0}^n)$: Takes as an input a secret key x of the delegator, a degree d and a topic description topic and sends $n+1$ evaluations E_i of the polynomial $p(T) = \kappa(xH_0 + xH_1T + ... + xH_{n-1}T^{n-1})$ to the delegatee, where H_i are the base points derived from DOOPS.GenerateTopic(d, topic). The delegation parameters D consist of these evaluations. For simplicity evaluation points can be derived deterministically by hashing the topic and the degree along with the index to the field $\mathbb{F}_r$. Note that these evaluations are exactly the signatures, therefore the delegatee can easily verify them using DOOPS.Verify.
- DOOPS.Sign$(D, m) \rightarrow \sigma$: Takes as input the delegation parameters D, a message m, and a topic. It derives vector of base points $\{H_i\}_{i=0}^d = \text{OOPS.GenerateTopic}(d, \text{topic})$. It uses the evaluations in D to interpolate the coefficients of the polynomial $p(T)$, which are κH_i and returns a signature $\sigma = \kappa(\sum_{i=0}^{n-1} m^i H_i) \in \mathbb{G}_1$.

6.2 Security proof

The security of the delegatable signature scheme in the AGM relies on the security of our signature scheme and the hardness of the discrete logarithm problem. Since the adversary only receives $n+1$ signatures that are essentially polynomial evaluations of polynomials with random coefficients in $\mathbb{F}_r$, and a different context has different random coefficients, they cannot derive the private key κ and therefore cannot sign messages outside the context of the delegated topic. Additionally, since the polynomial is of degree n only the party that either knows the coefficients of the polynomial or has at least $n+1$ evaluations can sign messages in that context. Thus, the delegated signature scheme ensures that signing rights

are securely delegated to the delegatee for a specific topic without revealing the private key of the delegator.

6.3 Additional observations

We note that this scheme is highly compatible with existing BLS signature implementations and can use them as a blackbox. Concretely, assume we have the signing BLS.Sign and hash to curve BLS.$\mathcal{H}$ functions, the delegatee can invoke BLS.Sign on the messages $m_i = (\text{topic} \parallel \deg \parallel i)$ which will output the signature $\sigma_i = x$ BLS.$\mathcal{H}(m_i)$. Observe that these signatures correspond to the coefficients of the polynomial which is used in the OOPS signature scheme. Having the coefficients, to sign a message m, delegatee then just computes an inner product between coefficients and powers of m.

This makes a gradual deployment of OOPS possible, even with existing hardware wallets and audited BLS implementations, while introducing a security tradeoff of being able to forge signatures that is only limited to a specific topic.

7 Oblivious transfer from blind polynomial evaluation

7.1 Blind polynomial evaluation

One of the primitives that we need for our oblivious transfer protocol is a blind polynomial evaluation [NP06]. This is a protocol involving two parties, a sender and a receiver. The sender holds a polynomial $p(T)$ and the receiver holds a secret value s. At the end of the protocol, the receiver should learn the evaluation $p(s)$ without revealing s to the sender and without learning anything else about the polynomial $p(T)$ beyond the evaluation $p(s)$. We show how to use our scheme to perform blind evaluations of polynomials, such that most of the work can be done ahead of the receiver choosing the evalutaion point. Let κ be a private key of a signer and let $[\kappa^{-1}]_2$ be the corresponding verification key. Let $p(T) = \kappa(C_0 + C_1T + C_2T^2 + ... + C_{n-1}T^{n-1})$ be a polynomial in $\mathbb{G}_1[T]$ and let the points $C_0, ..., C_n$ be public and such that discrete logarithm of each point with respect to a generator of $\mathbb{G}_1$ is unknown. The receiver picks a secret value s and computes an evaluation $p'(s) = C_0 + sC_1 + ... + s^{n-1}C_{n-1}$. Note that for a secret s, $p'(s)$ is a perfectly hiding commitment and additionally note that $p'(s) = \kappa^{-1}p(s)$. The receiver sends $p'(s)$ to the sender who then computes a signature on $p'(s)$, *i.e.* $\sigma = \kappa p'(s) = p(s)$ and sends it back to the receiver. The Receiver now has the evaluation $p(s)$ without revealing s to the sender and without learning anything else about the polynomial $p(T)$. Additionally, suppose that sender holds two polynomials $p_1(T)$ and $p_2(T)$. Since $p'(s)$ is a perfectly blinding commitment, the sender cannot distinguish whether it is evaluating $p_1(T)$ or $p_2(T)$ which is a crucial property for building efficient oblivious transfer.

7.2 Oblivious transfer

An 1-out-of-n oblivious transfer is a cryptographic primitive that allows a receiver to obtain one out of n messages from a sender without revealing its choice, while additionally not learn-

ing anything about the other messages [CO15]. OT is often used in multi-party computation protocols such as secure function evaluation, garbled circuits, and distributed key generation [Lyu+25]. 1-out-of-n oblivious transfer protocol involves two parties, a sender who holds n messages $m_1, ..., m_n$ and a receiver who wants to obtain one of the messages m_c for a choice $c \in \{1, ..., n\}$. At the end of the protocol, the receiver should learn m_c without revealing c to the sender and without learning anything about other messages m_i for $i \neq c$.

7.3 Construction overview

We build a 1-out-of-n oblivious transfer protocol from our blind polynomial evaluation scheme as follows. We achieve this by assigning a unique polynomial $p_i(T) \in \mathbb{G}_1[T]$ to each message. Then, without loss of generality, the encryption key is chosen to be a hash of polynomial evaluation at 0, *i.e.*, $k_i = \text{Hash}(p_i(0))$. For the application of oblivious transfer we can work with polynomials of degree 1, since this minimizes the communication overhead. The sender then encrypts each message m_i under the encryption key k_i and sends all the ciphertexts to the receiver. Coefficients of the polynomial depend on the private key of the sender as in Section 7.1, thus the receiver cannot compute evaluations on their own. To obtain the encryption key for a specific message, the receiver picks a secret value s, computes $p_i{}'(s)$, and then gets a blind evaluation $p_i(s)$ from the sender. From two different evaluations of $p_i(T)$, the receiver can interpolate the coefficients of the polynomial and compute $p_i(0)$ to derive the encryption key. Further, since the sender blindly evaluates one of the polynomials without knowing which, it cannot learn which message the receiver picked. Finally, since the receiver only learns one evaluation of the polynomial it cannot interpolate other polynomials and thus cannot learn anything about other messages.

Our oblivious transfer protocol has several benefits. It relies on standard cryptographic assumptions such as the hardness of the discrete logarithm problem and unforgeability of our signature scheme in the AGM. It is very efficient to compute for both sender and receiver, and most interestingly encryption keys do not depend on the receiver's choice. This means that all encryption keys, and therefore encrypted messages, can be pre-computed by the sender before the protocol even starts. This is useful in many scenarios, especially when one sender needs to interact with many receivers and when there are multiple oblivious transfers to be performed. Pre-computation significantly reduces the online computation required during the protocol execution since the sender can first publish all encrypted messages before the actual protocol starts.

7.4 Protocol

Setup.

1. $\text{Setup}(1^\lambda) \to (C_0, ..., C_{2n-1})$, where each $C_i \in \mathbb{G}_1$ and the discrete logarithm of each point with respect to a generator of $\mathbb{G}_1$ is unknown.

2. Sender and receiver agree on a cryptographic hash function $\text{Hash} : \mathbb{G}_1 \to \{0,1\}^l$ for some security parameter l.
3. Sender and receiver agree on an authenticated encryption scheme $\text{AE} = (\text{Enc}, \text{Dec})$.
4. Sender and receiver agree on an evaluation point $e \in \mathbb{F}_r^*$, at which the first evaluation will occur

Pre-computation.

1. Sender chooses a private key $\kappa \in \mathbb{F}_r^*$ and computes the corresponding public key $\text{pk} = [\kappa^{-1}]_2$
2. For each message m_i for $i \in \{0, ..., n-1\}$, the sender constructs a polynomial $p_i(T) = \kappa(C_2 i + C_{2i+1})T$.
3. For each message m_i, the sender computes an encryption key $k_i = \text{Hash}(p_i(0))$ and computes a ciphertext $\text{ct}_i = \text{Enc}(k_i, m_i)$.
4. For each polynomial $p_i(T)$, the sender computes a signature on the evaluation point e, *i.e.*, $\sigma_i = p_i(e)$.
5. Sender sends all ciphertexts $\text{ct}_0, ..., \text{ct}_{n-1}$ to the receiver.
6. Sender sends all signatures $\sigma_0, ..., \sigma_{n-1}$ to the receiver.

Online phase.

1. Receiver chooses $c \in \{0, ..., n-1\}$ and a secret value $s \in \mathbb{F}_r^*$.
2. Receiver computes $p'_c(s) = C_{2c} + sC_{2c+1}$ and sends it to the sender. This is a perfectly hiding evaluation.
3. Sender computes a signature $\sigma = [x]_1 p'_c(s)$ and sends it back to the receiver.
4. Receiver now has the evaluation $p_c(s)$. Using $p_c(e)$ received during pre-computation and $p_c(s)$ received from the sender, the receiver interpolates the coefficients of the polynomial to compute $p_c(0)$ and derives the encryption key $k_c = \text{Hash}(p_c(0))$.
5. Finally, the receiver decrypts the ciphertext ct_c using the encryption key k_c to obtain the message $m_c = \text{Dec}(k_c, \text{ct}_c)$.

7.5 Security proof

Sender privacy. The receiver only learns one extra evaluation, that of the polynomial $p_c(T)$ for their choice c. Since the polynomials are of degree 1, the receiver needs two evaluations to interpolate the polynomial and compute $p_c(0)$. Thus, the receiver cannot learn

anything about other messages beyond their choice. The security proof is identical to the one in 5.1.

Receiver privacy. Given the random scalar s chosen by the receiver, the point $p'_{c(s)}$ is indistinguishable from randomly sampled point in $\mathbb{G}_1$. Thus, the sender cannot distinguish which polynomial it is evaluating. Therefore, the sender learns nothing about the receiver's choice.

7.6 Additional observations

Using PIR. In order to decrypt a message, the receiver needs the decryption key and a ciphertext. To obtain the ciphertext the receiver cannot query the sender for the ciphertext since that would leak its choice. Since the sender publishes all encrypted messages before the protocol starts, the receiver can download all ciphertexts and then decrypt the one corresponding to its choice. This means that the receiver has to store all ciphertexts locally which can be a problem in memory constrainted environments. This can be avoided by using a private information retrieval (PIR) protocol [Cho+98] which allows a client to download a specific entry from a database without revealing its choice to the server. PIR doesn't guarantee that the receiver learns nothing about the database, but since the database consists of ciphertexts, which are hiding, the receiver anyway cannot learn anything about the other messages beyond its choice.

Two messages in the online round. Instead of agreeing on a single evaluation point e during setup, the receiver can pick two random values $s_1, s_2 \in \mathbb{F}_r^*$ and send two blinded evaluations $p'_c(s_1), p'_c(s_2)$ to the sender. The sender then computes signatures on both blinded evaluations and sends them back to the receiver. The receiver now has two evaluations of the polynomial $p_c(T)$ and can directly interpolate the polynomial to compute $p_c(0)$ and derive the encryption key. This way, the receiver doesn't need to store any signatures from the precomputation phase. However, this comes at the cost of a slightly increased communication overhead in the online phase and computation since now the sender needs to compute two signatures and the receiver needs to receive two signatures. Note that if the receiver tried to cheat and evaluated polynomials with different coefficients, they will learn nothing since they only learned at most one evaluation per polynomial. It's also straightforward to add efficient proofs of evaluation of a consistent polynomial.

DKG. We can build a non-interactive distributed key generation (DKG) protocol from OOPS by using very similar construction as in the oblivious transfer. Instead of allowing receiver to chose a message and a polynomial, sender can provably show that it sent a correct evaluation to each party. To avoid interaction sender can evaluate a polynomial at a public key of the receiver. More specifically, let $[t]_2$ be a public key of the receiver. Evaluation of polynomial $[xC_0, xC_1]$ at $[t]_2$ can be obtained by computing: $\langle xC_0, [1]_2\rangle + \langle C_1, x[t]_2\rangle$. To

ensure public verifiability TBD. The last issue is that receiver gets evaluation it $\mathbb{G}_t$ instead of $\mathbb{G}_1$. This can be solved by moving keys to $\mathbb{G}_t$, *i.e.*, encrpytion keys can be derived by hashing to $\mathbb{G}_t$ instead of $\mathbb{G}_1$. By obtaining 1 evaluation at public e and 1 evaluation at t only the receiver can interpolate $p(T)$ in $\mathbb{G}_t$ and compute $p(0)$ to derive the encryption key.

8 Verifiable partially oblivious PRFs

In an oblivious PRF protocol, a client wants a server to evaluate a PRF using the server's secret key, while the server doesn't learn the message the PRF was evaluated on. In a constrainted oblivious PRFs the client is only allowed to evaluate the PRF on a subset of the messages. If the topic is public, it's called a partially oblivious PRF. In a verifiable oblivious PRF, any third party can verify the correctness of the PRF evaluation.

Client → Server. The client chooses random $r_0, r_1 \leftarrow \mathbb{F}_r$ and prepares $T_0 = H_0 + r_0 H_1$ and $T_1 = H_0 + r_1 H_1$. The client sends T_0 and T_1. This is perfectly hiding, so the server doesn't learn the base points.

Server → Client. The server computes $K_0 = \kappa T_0$ and $K_1 = \kappa T_1$ and sends those to the client.

Client. The client takes the two signatures and, using these two evaluations of the polynomial with coefficients T_0 and T_1, can evaluate it at any message m, by finding the linear combination resulting in $K_m = \kappa(T_0 + mT_1)$. Note that only the client can perform this, since nobody else knows r_0 and r_1. The correctness of the evaluation can be checked by $\langle T_0 + mT_1,\ \text{pk}\rangle = \langle K_m, G_2\rangle$.

Note that you can only evaluate the PRF securely once, since one more evaluation will allow anyone to evaluate the PRF themselves. The constraint comes in the form of the client choosing specific base points, which can be derived, *e.g.*, from a known message. If the client wants the topic to be known to the server, they can reveal the base points to the server in the first step.

8.1 Security proof

Client privacy. Since the client blinds the base points using random scalars r_0 and r_1, points T_0 and T_1 are indistinguishable from random points in $\mathbb{G}_1$. Thus, the server cannot learn anything about the base points and therefore cannot learn anything about the message the PRF is evaluated on.

OPRF security. Assume there is an algorithm that evaluates the OPRF without getting two valid singatures from the server. Then, we can use this algorithm to build an adversary that forges a signature for OOPS by breaking the OPRF protocol using the algorithm.

8.2 Additional observations

Threshold OPRF. The OPRF scheme is easily thresholdizable using secret sharing schemes for field elements.

Higher degree verifiable partially oblivious PRFs. It's straightforward to allow more public evaluations of the PRF by increasing the degree of the polynomial.

9 Future work

OOPS-zk-creds Following the observations in [Ros+23] and [CFQ19], it is possible to have the prover, wishing to show an identity is a member of the current set of identities, to efficiently output a hiding Pedersen commitment P to sk, the identity's secret key. To implement ShowCred for a topic and message m, the prover performs the following:

1. Provide a discrete logarithm equality proof between P and $\text{nullifier} = \text{sk} \cdot \mathcal{H}(\text{topic} \parallel d \parallel \text{pk})$, a fresh base point on $\mathbb{G}_2$.
2. Provide a discrete logarithm equality proof between P and $\text{pk}_{\text{enc}} = \text{sk} \cdot (G_1 + H_0)$, a hiding commitment to the identity's known public key, where H_0 is the base point obtained from OOPS. GenerateTopic.
3. Output $\sigma = \text{OOPS. Sign}(\text{sk}, m, d, \text{topic})$.
4. Verification is adapted to be according to the new base: $\langle \sigma, \mathcal{H}(\text{topic} \parallel d \parallel \text{pk}) \rangle = \langle M, \text{ nullifier} \rangle$, where M is the message constructed as in OOPS.
5. If $n + 1$ signatures are revealed, compute $\sigma_0 = \text{sk} \cdot H_0$, which is the signature on the message 0 and reveal $\text{pk} = \text{pk}_{\text{enc}} - \sigma_0$. Note that this is not post-quantum resistant. The details of the discrete logarithm equality proofs are left for future work.

10 Acknowledgements

We thank Nicolas Mohnblatt for his feedback and valuable comments. The initial direction for OOPS arose in an in-progress work with Guillermo Angeris. We thank ChatGPT for helpful suggestions.

Bibliography

[Ros+23] M. Rosenberg, J. White, C. Garman, and I. Miers, "zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 790–808.

[Whi19] B. WhiteHat, "Semaphore RLN, rate limiting nullifier for spam prevention in anonymous p2p setting." Feb. 2019.

[Int25] "Privacy Pass Working Group." [Online]. Available: https://datatracker.ietf.org/wg/privacypass/about/

[GHW23] D. Grandjean, L. Heimbach, and R. Wattenhofer, "Ethereum Proof-of-Stake Consensus Layer: Participation and Decentralization." [Online]. Available: https://arxiv.org/abs/2306.10777

[Bon+09] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a linear subspace: Signature schemes for network coding," in *International workshop on public key cryptography*, 2009, pp. 68–87.

[BDN18] D. Boneh, M. Drijvers, and G. Neven, "BLS multi-signatures with public-key aggregation," *URL: https://crypto. stanford. edu/dabo/pubs/papers/BLSmultisig. html*, 2018.

[AN24] G. Almashaqbeh and A. Nitulescu, "Anonymous, timed and revocable proxy signatures," in *International Conference on Information Security*, 2024, pp. 23–43.

[Jar+17] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu, "TOPPSS: Cost-minimal Password-Protected Secret Sharing based on Threshold OPRF." [Online]. Available: https://eprint.iacr.org/2017/363

[JKR19] S. Jarecki, H. Krawczyk, and J. Resch, "Updatable Oblivious Key Management for Storage Systems." [Online]. Available: https://eprint.iacr.org/2019/1275

[CO15] T. Chou and C. Orlandi, "The simplest protocol for oblivious transfer," in *International Conference on Cryptology and Information Security in Latin America*, 2015, pp. 40–58.

[Kol+16] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient Batched Oblivious PRF with Applications to Private Set Intersection." [Online]. Available: https://eprint.iacr.org/2016/799

[BBS04] D. Boneh, X. Boyen, and H. Shacham, "Short Group Signatures." [Online]. Available: https://eprint.iacr.org/2004/174

[TZ23] S. Tessaro and C. Zhu, "Revisiting BBS Signatures." [Online]. Available: https://eprint.iacr.org/2023/275

[ASM08] M. H. Au, W. Susilo, and Y. Mu, "Constant-Size Dynamic k-TAA". [Online]. Available: https://eprint.iacr.org/2008/136

[CDL16] J. Camenisch, M. Drijvers, and A. Lehmann, "Anonymous Attestation Using the Strong Diffie Hellman Assumption Revisited." [Online]. Available: https://eprint.iacr.org/2016/663

[BFL20] B. Bauer, G. Fuchsbauer, and J. Loss, "A classification of computational assumptions in the algebraic group model," in *Annual International Cryptology Conference*, 2020, pp. 121–151.

[NP06] M. Naor and B. Pinkas, "Oblivious polynomial evaluation," *SIAM Journal on Computing*, vol. 35, no. 5, pp. 1254–1281, 2006.

[Lyu+25] Y. Lyu, Z. Li, H.-S. Zhou, and X. Deng, "Threshold ECDSA in Two Rounds." [Online]. Available: https://eprint.iacr.org/2025/1696

[Cho+98] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, Nov. 1998, doi: 10.1145/293347.293350.

[CFQ19] M. Campanelli, D. Fiore, and A. Querol, "LegoSNARK: Modular design and composition of succinct zero-knowledge proofs," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2075–2092.