An ECDSA Nullifier Scheme for Zero Knowledge Proofs

by

Aayush Gupta*, Kobi Gurkan**

Abstract

ZK-SNARKs (Zero Knowledge Succinct Noninteractive ARguments of Knowledge) are one of the most promising new applied cryptography tools: proofs allow anyone to prove a property about some data, without revealing that data. Largely spurred by the adoption of cryptographic primitives in blockchain systems, ZK-SNARKs are rapidly becoming computationally practical in real-world settings, shown by i.e. tornado.cash and rollups. These have enabled ideation for new identity applications based on anonymous proof-of-ownership. One of the primary technologies that would enable the jump from existing apps to such systems is the development of deterministic nullifiers.

Nullifiers are used as a public commitment to a specific anonymous account, to forbid actions like double spending, or allow a consistent identity between anonymous actions. We identify a new deterministic signature algorithm that both uniquely identifies the keypair, and keeps the account identity secret. In this work, we will define the full DDH-VRF construction, and prove uniqueness, secrecy, and existential unforgeability. We will also demonstrate a proof of concept of the nullifier.

Thesis Supervisor: Vinod Vaikuntanathan (for Aayush's thesis)
Title: Professor

* = 0xPARC, MIT
** = Geometry Research

Acknowledgments

Thanks to Lakshman Sankar for guidance, especially for helping answer questions on security and validating the direction on the day-to-day, and helping me stay accountable via meetings and check-ins.

Thanks to Kobi Gurkan for helping me think through the nitty-gritty of the proofs, especially weekend calls to explain the details of the papers, pointing to useful papers and resources, and basically coming up with the research direction.

Thanks to Wei Jie Koh for helping to write a formal specification of my Rust code to help production adoption.

Thanks to Vivek Bhupatiraju for helping me whiteboard details of the trickier proofs.

Thanks to Remco and Kobi for helping put together the full picture of the quantum insecurity of the algorithm.

Thanks to Nalin, Adhyyan, and Brian Gu for helping to come up with the original Stealthdrop construction that motivated this direction, and for brainstorming for days on possible solutions to address this problem.

Thanks to Justin Glibert for insightful discussion and piercing questions for each topic I considered along the way.

Thanks to Jordi Baylina and Blaine Bublitz for support with circom and snarkjs toolstacks.

Thanks to Prof. Yael Kalai for helping to review this paper.

Thanks to Prof. Dan Boneh and Riad Wahby for helping to validate the original research direction and suggesting some great changes early.

Thanks to Prof. Vaikuntanathan for approving my proposal and thesis, and helping validate the research direction.

Contents

Chapter 1

Introduction

Due to the widespread adoption of the Ethereum Virtual Machine (EVM) by various cryptography research organizations and cryptocurrency platforms, ECDSA signatures and keypairs are now commonplace. As a result, porting existing primitives to the pairing-unfriendly secp256k1 elliptic curve is desirable for widespread research adoption.

Spurred by the adoption of cryptocurrency, the ability to reveal specific parts of your identity in lieu of the whole is an interesting new primitive [9]. Signatures via ECDSA are nondeterministic, meaning a signer can produce an arbitrary number of signatures for a message. However, they do provide non-repudiation of signed data, and enable applications such as semi-anonymous message boards.

For instance, one can prove via a zero knowledge proof of knowledge (ZKP) that their account satisfies some property i.e. they are an NFT holder or protocol user, without revealing who they are. Set membership proofs are a typical usecase of zk-snarks for privacy [8], and we can imagine a zk proof of the form "I can prove that I own the private key [via generating a valid signature] for some public key that is a leaf of the merkle tree comprised of all set members, with this public merkle root." For such applications, a person merely needs to prove the existence of at least one valid signature per message to be sure that such a message is legitimate [19]. However, such applications have the advantage that there is no uniqueness constraint on the provers: that is, the same wallet proving itself as a member multiple times is

intended behavior. However, many applications require a maximum of one action per user, especially protocols that desire sybil resistance like claiming an airdrop.

For a concrete example, a zero knowledge airdrop [26] requires that an anonymous claimer can produce some unique public identifier of a claim to a single address on an airdrop, that does not reveal the value of that address[35]. One can imagine a "claimer" can send a zk-proof of knowledge of merkle path to some public merkle root, along with a proof of private key ownership (whether signature verification or something else). A public nullifier signal ensures they cannot claim again in the future. This unique public identifier is coined a "nullifier" because the address nullifies its ability to perform the action again. For a specific nullifier function, a hash of the public key would be ideal but can easily be brute forced due to the finite number of on-chain addresses, so alternate solutions are needed. One can imagine that a signature would be an ideal nullifier; however, most signature algorithms (and all presently deployed ones on Ethereum) have 2^{256} valid signatures for the same message and public key, so you can imagine someone can "prove" membership an effectively infinite number of times and have a different "nullifier" each time, defeating the purpose of a unique nullifier.

One can then imagine that hash(message, secret key) is a decent nullifier, where each app has (usually) one canonical message – there's no way to reverse engineer the secret key, it's lightweight and computable in a hardware wallet, and it is unique for each account and application. However, we can't verify it without access to the secret key itself. For security reasons, we want a way to be able to do all of these computations without a user having to insert their private key anywhere, especially not copy paste it as plaintext. For anything that does need a private key, we want computation to be very lightweight so we can run it on a hardware wallet as well if needed: the complex elliptic curve pairing functions required to prove ZK SNARKs are not feasible to compute in current memory-constrained hardware wallets, and because ZK proof systems evolve so rapidly, we don't want wallets to commit to a specific system this early. However, this simple construction provides the key insight that we use to motivate our nullifier scheme.

If we want to forbid actions like double spending or double claiming, we need these nullifiers to be unique per account. Because ECDSA signatures are nondeterministic, signatures don't suffice; we need a new deterministic function, verifiable with only the public key. We want the nullifier to be non-interactive, to uniquely identify the keypair yet keep the account identity secret. Overall, the key insight is that such nullifiers can be used as a public commitment to a specific anonymous account.

1.1 Contributions

We implement and verify a method for a non-interactive nullifier scheme. We also provide an explanatory blog post with intuitions for layfolk.

The exact definition of a successful nullifier is as follows: we want a deterministic nullifier function N(sk, pk), for an ECDSA keypair (sk, pk), to have the following properties, for some application-specific message m, which can also be considered to encapsulate information about any common reference strings. Note that $proof$ can be any proof transcript at all, including an adversarially chosen one. ϵ is any negligible function. Note that these definitions may have more inputs as well, such as common reference inputs. We also assume that no address has public key 0 for ease of notation.

1. **Correctness**. For all pk_i, and verification function Ver, $Ver(N(m, sk_i, pk_i),\ proof_j, m, pk_i) = pk_i$, for all such $proof_j$, and $P[V(x, x \neq N(m, sk_i, pk_i) \forall i,\ proof) = 0] > 1 - \epsilon$. Note that Ver does not take any private input, thus it must be possible to verify the signature scheme with only public information.

2. **Uniqueness**. Any keyholder cannot generate two nullifiers. Formally, $V[x, proof_x] = pk_1, V[y, proof_y] = pk_1)$ implies $x = y$. This implies the function must be deterministic.

3. **Secrecy**. Also called unpredictability/hiding [18]. An adversary $\mathcal{A}$ with pk_i, a succinct list of all valid public keys, as well as pk_j, a list of all public keys approved by the set inclusion algorithm, cannot distinguish two keys

given the nullifier, the only public ZK proof output. For $pk_1, pk_2 \in \{pk_j\}$, $P\left(\mathcal{A}[N(sk_1, pk_1), N(sk_2, pk_2), \{pk_j\}] = (pk_1, pk_2)\right) < 0.5 + \epsilon$.

Note these definitions are almost identical to VUFs [18], but all current implementations of VUFs are not compatible with the secp256k1 curve. Note that it will usually be desirable to keep the public key private using this nullifier, so we intend the verification step to occur within a zk-proof. Our definitions can be achieved via DDH-VRFs that incorporate ZK-SNARK verifiers instead of classical sigma protocols [6], since proving public key set membership is much easier in ZK-SNARKS.

1.2 Applications

Standardizing such a nullifier scheme into at least one wallet would make any applications that rely on unique, anonymous identities practical. These include airdrops, persistent identities on message boards, and uniqueness claims for proof of ownership.

ZK airdrops are achieved via publishing a set accumulator (i.e. a Merkle tree) of allowed users and having users submit a proof to claim an airdrop from an unlinked account. Eligible people would prove proof of private key ownership via signature, along with a proof that the corresponding public key is in the Merkle tree by providing a Merkle path, and in the same proof verify their nullifier as a check for double-claiming.

Message boards can have a persistent anonymous identity, meaning someone can post under the same nullifier multiple times, and everyone knows that they are the same person, but not who that person is. Users would simply upload a nullifier where the message corresponds to the unique ID of the thread, along with an ECDSA signature from their public key in the same ZK-SNARK of the message contents and timestamp where the signature verification is kept private in the SNARK.

We can also build a more secure version of tornado.cash, where instead of leaking the nullifier string in plaintext on the frontend from a server via the website, the nullifier can simply be this ECDSA nullifier corresponding to some message in a standard format, like “tornado.cash note 5 for 10 eth”.

We think that wallets that adopt this standard first will hold the key for their users being able to interact with the next generation of ZK applications first. We hope this standard becomes as commonplace as ECDSA signing, within every secure enclave.

1.3 Related Work

This specific problem has only recently become of particular interest due to the aforementioned zero knowledge applications reaching production. Nullifiers have used since zcash [4] and tornado.cash [2], but we have a non-interactivity requirement that they may not. Without a non-interactivity requirement, a user could simply hash a random string and use another hash of that preimage as their nullifier from then on: tornado.cash uses a similar scheme based on random strings. However, we want to be able to publish a nullifier without any such interaction, so it is much harder to build off of their constructions.

It is also not possible to create an algebraic nullifier function using only one group and it's algebraic operations, as proven in [1]. However, using tools such as MPC or pairings may allow us to map between multiple fields/groups and thus not be solely algebraic. In addition, treating a hash function as a pseudorandom oracle also breaks algebraic constraints, as the output of a hash function cannot be expressed as a linear combination of its inputs.

There is also research on fair ZK, in which a unique witness can be made to have a deterministic ZK proof, and thus a hash can be used as a nullifier [23]. However, this is not presently useful, as we still need a deterministic nullifier to prove we know the private key. Note that our construction is also similar to a unique ring signature over ECDSA, but where each public key, for each set, emits a unique signature that doesn't identify them [29].

One proposed solution is using MPC to shuffle an encrypted secret, first proposed by Riad Wahby. This solution would allow us to have deterministic nullifiers with guarantees up to a group instead of an individual; users would simply add their

nullifiers to an anonymity set, which would be shuffled in such a way that we can detect if someone in the group is interacting twice with the same application, but we can't pinpoint which one. This would work by first having a coordinator publish encrypted secrets, and future participants "shuffle" which messages are encrypted by which addresses, leading to a 1-of-n trust guarantee (i.e. at least one shuffler must be honest for it to be unlinkable). This primarily works with RSA-style encryptions and does not seem practical for our use case, due to the interactivity required to generate a valid RSA key pair ahead of time. Verifiable random shuffles also depend on the existence of a homomorphic encryption function, which the standardized Metamask ed25519 encryption algorithm ($X25519\_XSalsa20\_Poly1305$ is standard, with a MAC and with keypairs derived from ECDSA private keys) is not. Note this is impractical for our setting, as ECDSA has no associated encryption/decryption function (except for ECIES, which relies on a Diffie-Hellman like setup).

There is additionally another body of work on verifiable unpredictable functions (VUFs), which has the same zero knowledge and security guarantees as the solution we are looking for [18]. The output of some VUF would be used to generate either encrypted randomness for a user that only they can decrypt, or as the basis for a deterministic signature. However, research is scant especially compared to VRFs (verifiable random functions). The algorithm outlined in the VUF paper [18] is promising but requires a pairing friendly curve to verify, similar to BLS. These are usually groups where decisional Diffie-Hellman (DDH) is easy, but computational Diffie-Hellman (CDH) is hard. However, ECDSA does not operate on the same elliptic curve, and DDH is hard so it is pairing unfriendly. However, exploring other pairing based schemes could be promising in the future.

This leaves one final avenue for us to proceed on. We want a deterministic function of a user's secret key, that can be verified with only their public key, eventually done inside a ZK-SNARK to keep the public key itself anonymous behind a set membership check. We choose to implement and standardize such a deterministic signature scheme, specifically building atop a concrete DDH-VRF scheme [6], such as EC-VRF [15] as originally described in a paper about NSEC5 in DNSSEC [27].

1.4 Roadmap

We first provide definitions of all the relevant existing cryptographic primitives needed to understand the nullifier construction.

We then present the nullifier construction of the BLS-like [5] signature scheme. The construction is based on section 3.2 of [10] and BLS, and is effectively fixed Goh-Jarecki signatures where we subsitute r inside the hash with pk instead to make it deterministic [14]. We prove all the relevant nullifier requirements.

Finally, we present moral considerations and present benchmarks, as well as describe future work.

Chapter 2

Preliminaries

In this section, we will introduce the hashing primitives, signature algorithms, ZK-SNARK systems, and internet infrastructure systems that our construction relies on.

2.1 Hashing

Our nullifier algorithm relies on several hashing algorithms, including SHA256, SHA512, and the IETF RFC standard hash to curve algorithm for secp256k1 QUUX-V01-CS02-with-secp256k1_XMD:SHA-256_SSWU_RO_ (we will abbreviate this last algorithm as hash2curve henceforth, for brevity) [12].

There are many SNARK-friendly hash functions, like MiMC, Poseidon, or Pederson. However, we didn't use those in our initial proof of concept and paper, because we want to guarantee maximum compatibility and probability of adoption by wallets. If a prominent wallet provider indicates a willingness to use one of these hash functions, we believe it is advantageous to switch our scheme to use it.

Note that our construction also relies on a hash to curve algorithm, because a hash multiplied by the generator would break the existential unforgeability of any signature scheme dependent on it [33].

We assume these hash algorithms are collision resistant and deriving a preimage is hard given the hashed value, and breaking each requires 2^n queries for some n.

2.2 Signatures

2.2.1 ECDSA

ECDSA is the signature protocol used by Bitcoin, Ethereum and most blockchain systems [25, 7], due to both Schnorr's copyright and ECDSA's relatively smaller key size, especially when compared to RSA. Most RSA keys are 2,048 bits, but the much shorter 256-bit ECDSA key provides roughly equal security to a 3,248 bit RSA key [28].

ECDSA uses the secp256k1 curve, meaning all the points are on $y^2 = x^3 + 7$ [20]. Because almost all existing blockchain and public key infrastructure uses this curve for non-deterministic signatures, we are interested in a nullifier construction for this class of curves specifically.

2.2.2 Pairing Friendly Signatures

BLS is a deterministic proof over pairing-based curves. One notable thing about secp256k1 is that it is not pairing friendly. One clear reason for this (of many) is that decisional Diffie Hellman and computational Diffie Hellman are both hard in secp256k1; but, in pairing friendly curves such as the ones used in BLS, decisional Diffie Hellman is easy, while computational Diffie Hellman is still hard [5]. Note that this means that a simple deterministic signature scheme (which would give us our ideal nullifier for free) such as BLS will not work with ECDSA, although if we use its form as an inspiration [5].

2.2.3 Other Signatures

Note that RSA signatures are deterministic, which means that those signatures could directly be proven in the ZK-SNARK – we wouldn't need a bespoke signature scheme.

Verifiable Unpredictable Functions, or VUFs, are another pairing-curve based construction that allows specific construction of nullifier-style identities [18] and are impractical for ECDSA largely due to the reliance on the pairing check.

2.3 Ethereum

Ethereum is a data available public ledger that can act as a decentralized verification layer for our contracts and SNARKs. There are several types of wallets, including secure enclave enabled hardware wallets, software wallets that run on the OS level, and browser wallets that can query any secure enclaves required and directly interact with smart contracts via web interfaces. We want to isolate simple operations that leak the secret key to the secure enclave, isolate the resulting signals that leak anonymity within the secure wallet application or a client-side-only web app, and only reveal the ZK-SNARKed public data to the world. To ensure data availability and censorship resistance, ideally this data is hosted via a credibly neutral blockchain layer that can act as a 3rd party verifier, trusted via decentralization.

2.3.1 Constraints

We want a nullifier algorithm to be practical for both in-browser wallets such as Metamask and hardware wallets such as Ledger and Trezor. However, especially hardware wallet environments are considerably resource-constrained: the inexpensive secure enclave chips can only compute limited functions, and the smallest ones have a total of 320 kB of flash memory for all applications [34]. Thus, it will take impractical amounts of time (and is currently infeasible with that amount of memory) to compute a function as complex as a SNARK proof.

This constraint is the reason that we cannot generate a simple ZK-SNARK proof of knowledge of a private key that generates a valid public key, with the anonymity set as the only public input to the circuit. Even as memory limits increased, we do not want to commit secure enclaves to a specific proof system or proof, which would block applications from adding bespoke checks to the proof or upgrading the proving system. Thus, we opt to separate the calculation of signals in the enclave, from the verification of the signals in a ZK SNARK outside the enclave.

Note that we prefer to use a ZK-SNARK instead of a non-interactive Sigma protocol for ZK, because we want to additionally allow applications to easily add checks

inside the zero knowledge proof – for instance public key set membership in a Merkle tree, or an proof of a path in the Ethereum trie of on-chain state at some point for that public key.

2.4 ZK SNARKs

There are several generic proving systems, including Groth16 which uses R1CS arithmetization [3], PLONK which uses AIR [13], and Nova which uses relaxed R1CS [21]. We write our proof of concept in circom, which uses r1cs and groth16, although we expect to easily be able to swap out the groth16 proof for a Nova proof to boost performance. Note that we desire to use a statistical zero knowledge argument system over a computational one, to ensure that data remains zero knowledge after 256 qbit quantum supremacy (although the nullifier breaks past zero knowledge, which we comment on later).

We use a ZK-SNARK for easy composability with other algorithms, like a set inclusion algorithm for the public key, or any other existing circuit, along with the ease of verification of the succinct proof on-chain.

Chapter 3

Nullifier Scheme

3.1 Construction

This is a description of how the different entities of the secure enclave chip, browser wallet, client-side-only zk prover, and blockchain play their part in calculating the signatures. We assume the secure enclave is the only entity with access to the secret key sk and secure randomness r, and can be either a hardware enclave or a software enclave. We use exponential notation instead of multiplicative notation so a reader can apply their intuitions about the discrete log problem. In practice, hash_2 is a standard multi-value hash function like SHA256, and hash refers to a hash-to-curve function.

The enclave calculates and outputs publicly (to become a public input into the ZK SNARK):

$$nullifier = \text{hash}[m, pk]^{sk}$$

$$s = r + sk * c$$

The enclave also calculates and outputs this privately to the client (to become a private input into the ZK SNARK, as they de-anonymize the user):

$$c = \text{hash}_2\left(g, g^{sk}, \text{hash}[m, pk], \text{hash}[m, pk]^{sk}, g^r, \text{hash}[m, pk]^r\right)$$

$$pk = g^{sk}$$

$$g^r \qquad \text{[optional output]}$$

$$\text{hash}[m, pk]^r \qquad \text{[optional output]}$$

The optional inputs are included here to increase prover efficiency via precomputation, but they can be calculated from the other signals. The ZK SNARK, which is generated on a client-side only software that the prover runs, proves:

$$\leftarrow nullifier, s, c, pk, g, m$$

$$c == \text{hash}_2\left(g, pk, \text{hash}\left[m, pk\right], nullifier, \frac{g^s}{pk^c}, \frac{\text{hash}\left[m, pk\right]^s}{nullifier^c}\right)$$

$$pk \in set$$

Note that divisions as written are calculated in practice as subtractions, and that under the random oracle model for hashing, this check is implicitly proving the following (which you can easily verify completeness for by substituting $s = r + sk * c$).

$$\frac{g^s}{pk^c} = g^s / \left(g^{sk}\right)^c = g^r$$

$$\frac{\text{hash}\left[m, pk\right]^s}{nullifier^c} = \text{hash}\left[m, pk\right]^s / \left(\text{hash}[m, pk]^{sk}\right) = \text{hash}[m, pk]^r$$

We expect that users will include other application-specific checks as well in the same ZK SNARK, such as set inclusion checks for the stated public key (this can be done with a Merkle proof for a public Merkle root, for instance). For this circuit specific ZK SNARK, a verifier with the corresponding prover code will be posted on a Turing-complete public blockchain ledger, to allow anyone to both verify that the proof is accurate, and provide proofs that, upon verification, allow access to some specific on-chain action.

We use a ZK-SNARK instead of a simple ZK interactive protocol with Fiat-Shamir to allow application creators to easily extend the existing proof to fit their use cases.

3.2 Proofs

We prove security in the random oracle model. Specifically, we prove uniqueness (each public key can only generate one nullifier), secrecy (one cannot learn the public key from the public signals), and existential unforgeability (given proof signals, one cannot forge another proof or learn the secret key). Previous work has proved uniqueness, pseudorandomness, and collision resistance of the original signature scheme without the zero knowledge component [27].

3.2.1 Definitions

We define EUF via definitions similar to the BLS paper [5] and Goh-Jarecki's EDL paper [14], but for the secp256k1 curve.

Definition 3.1 (Group Definition) *If the group satisfies τ, t', ϵ', such that it τ is the ratio of timesteps needed to break DDH vs do the group operation (time = 1), and that an adversary can break CDH with probability ϵ' in t' timesteps. This group has order p.*

We assume for the implementation that this elliptic curve is secp256k1.

Definition 3.2 (Advantage of existentially forging) *This is the probability that an adversary can produce a nullifier and valid zk proof that validates for a known message m. Because this is an app specific, app-chosen parameter, we assume the adversary can choose this message m. We assume the hash function treated as random oracle, and we account for hash collision failure probabilities in our calculation.*

Definition 3.3 (Scheme security) *If an adversary in time t, with q_h queries to hash function, and q_s queries to sig oracle, can achieve an advantage of existentially forgery at most ϵ, then it is (t, q_h, q_s, ϵ) secure.*

Definition 3.4 (Security against Existential Forgery) *No forger can (t, q_h, q_s, ϵ) break a nullifier.*

Theorem 3.5 (Security Bound) *We will prove that* $t \leq t' - 2c_A(lgp)(q_H + q_S)$ *and* $\epsilon \geq 2e \cdot q_S\epsilon'$*, and* c_A *is a small constant between* 1 *and* 2*. Here* e *is the base of the natural logarithm.*

Theorem 3.6 (Computational Diffie Hellman Security) *A probabilistic algorithm* $\mathcal{A}$ *is said to* (t, ϵ)*-break* CDH *in a group* $G_{g,p}$ *if on input* (g, p, q) *and random query* $\left(g^a, g^b\right)$*, and after running in at most* t *steps,* $\mathcal{A}$ *computes* g^{ab} *with* ϵ *probability. We say that group* $G_{g,p}$ *is a* (t, ϵ)*-CDH group if no algorithm* (t, ϵ)*-breaks* CDH *in* $G_{g,p}$*.*

Note that these definitions are almost identical to BLS's signature security proof [5], and CDH follows because it is merely a reduction of discrete log [24].

Note that there is additionally a proof of set inclusion property: that one can a valid proof iff the public key is in the allowed set. Note that if we prove the uniqueness, secrecy, and existential unforgeability of the signature scheme, then this set inclusion proof immediately follows from the accumulator scheme check being correct, so we omit the formal proof.

3.2.2 Proof of Uniqueness

We prove that an adversarial secret key holder cannot generate more than one valid nullifier. Formally adversary $\mathcal{A}$ breaks uniqueness if:

$$Pr\left(\begin{array}{l} Gen(secp256k1) \rightarrow sk, pk \\ \quad sk, pk, g, m \rightarrow nullifier, s, c \\ \quad\quad c = \text{hash}_2\left(g, pk, \text{hash}\left[m, pk\right], nullifier, \dfrac{g^s}{pk^c}, \dfrac{\text{hash}\left[m, pk\right]^s}{nullifier^c}\right) \\ \quad\quad \mathcal{A} \leftarrow sk, pk, g, m \\ \quad\quad \mathcal{A} \rightarrow nullifier_{\mathcal{A}}, s_{\mathcal{A}}, c_{\mathcal{A}} \\ \quad\quad\quad nullifier_{\mathcal{A}} \neq nullifier \\ \quad\quad c_{\mathcal{A}} = \text{hash}_2\left(g, pk, \text{hash}\left[m, pk\right], nullifier_{\mathcal{A}}, \dfrac{g^{s_{\mathcal{A}}}}{pk^{c_{\mathcal{A}}}}, \dfrac{\text{hash}\left[m, pk\right]^{s_{\mathcal{A}}}}{nullifier_{\mathcal{A}}^{c_{\mathcal{A}}}}\right) \end{array}\right) > negl$$

where $negl$ is the negligible function. We can assume all inputs to hash_2 are well formed because we can add those constraints to the ZK-SNARK. We show that in the random oracle model, this probability is negligible.

Proof. Imagine the adversary makes q_h queries to the hash function. For each query, they have to commit to some $\left(pk, nullifier_{\mathcal{A}}, \frac{g^{s_{\mathcal{A}}}}{pk^{c_{\mathcal{A}}}}, \frac{\text{hash}[m,pk]^{s_{\mathcal{A}}}}{nullifier_{\mathcal{A}}^{c_{\mathcal{A}}}}\right)$ ahead of time to find $c_{\mathcal{A}}$. We can assume $pk = g^{sk}$ and $nullifier_{\mathcal{A}} = \text{hash}[m, pk]^{r'}$ for some r', since $\text{hash}[m, pk]$ is a generator of a cyclic prime group. Then the adversary has committed to $\left(g^{sk}, \text{hash}[m, pk]^{r'}, g^{s_{\mathcal{A}} - sk \cdot c_{\mathcal{A}}}, \text{hash}[m, pk]^{s_{\mathcal{A}} - r' \cdot c_{\mathcal{A}}}\right)$ ahead of time. Since they have committed to the exponents in the last two terms, the adversary has committed to an arbitrary $(sk - r')c_{\mathcal{A}}$ before finding out $c_{\mathcal{A}}$. This means that either $sk = r'$ and the discrete logs match so $nullifier_{\mathcal{A}} = nullifier$, or else $c_{\mathcal{A}}$ is 0, which happens with negligible probability $1/2^k$.

3.2.3 Proof of Existential Unforgeability

We prove that an adversary $\mathcal{A}$ with a (t, q_h, q_s, ϵ) advantage on breaking existential unforgeability for the signature scheme, can then (τ, ϵ) break CDH [11] in the random oracle model. Note that this proof is essential from the perspective of the secure enclave, to ensure that the data it produces does not leak information about the secret key to ZK-SNARK generator, which might be inside a wallet.

Proof. Imagine Alice wants to break CDH of (g, g^a, g^b), and Bob is the adversary who can generate a valid $(hash[m, pk], nullifier, s, c, m)$ tuple for a previously unseen message m after q_s queries of public and private outputs of the signature oracle (previous $(hash[m, pk], nullifier, s, c, m)$ tuples). To do this, Alice generates a random bit $b_i \sim Bernoulli(p)$ for the $\mathcal{A}$'s ith query of the signature oracle. This bit will control whether Alice predicts that Bob will try to break CDH on this round or not, and thus Alice will tailor their responses accordingly.

If $b_i = 1$, when Bob queries pk, Alice returns g^b. When Bob queries $\text{hash}[m, pk]$, Alice as the random oracle returns the value g^a. If Bob chooses to break the signature scheme this round and responds, because of the uniqueness proof, they return the nullifier $\text{hash}[m, pk]^{sk} = g^{ab}$ with probability ϵ. Alice can then use this value to break

Diffie Hellman. If Bob queries Alice for the nullifier, this scheme fails.

If $b_i = 0$, when Bob queries pk, Alice returns g^{r_i}, where r_i is uniformly random over the order of the prime field. When Bob queries hash$[m, pk]$, Alice as the random oracle returns the value g^{q_i}, where q_i is similarly uniformly randomly over the order of the prime field. If Bob queries Alice for the nullifier, Alice returns $g^{r_i \cdot q_i}$. If Bob tries to break the signature scheme by returning a tuple, this scheme fails because Alice gets no information.

Note that Alice needs to correctly guess all of the b_i's correctly to successfully answer Bob's first q_h queries as well as the final breaking query. Thus, similar to BLS, the probability of breaking the scheme is $p^{q_h}(1-p)\epsilon$. Maximizing p via derivative shows the max probability is attained at $p = \frac{q_h}{1+q_h}$, putting success probability at $\left(\frac{q_h^{q_h}}{(1+q_h)^{1+q_h}}\right)\epsilon$. For large q_h, using the identity $\lim_{x\to\infty}(1-\frac{1}{x})^x = e^{-1}$ gives that the probability is around $\frac{\epsilon}{e \cdot q_h}$ where e is the natural logarithm, and the number of steps is $t' = t + 2c_{\mathcal{A}}(lgp)(q_H + q_S)$, where q_H is the number of hash function queries, q_S is the number of signature queries, lgp is the approximate complexity of an exponentiation in a group of size p and dominates algorithm time per hash/signature, and c is an overhead constant that in practice is probably between 1 and 2.

Thus, if G is a (τ, t', ϵ')-CDH group, then there can exist no algorithm that (t, q_H, q_S, ϵ)-breaks our nullifier signature scheme with $t \leq t' - 2c_A(lgp)(q_H + q_S)$ and $\epsilon \geq 2e \cdot q_S\epsilon'$, and c_A is a small constant between 1 and 2. Note that this proof is almost the same as the BLS proof [5].

3.2.4 Proof of Secrecy

We prove that given only public signals, an adversary who is not the prover cannot learn anything about the original user's identity. That is, adversary $\mathcal{A}$ breaks secrecy if they can distinguish which public key was used to generate the nullifier. Formally,

$$Pr\left(\begin{array}{l} Gen(secp256k1) \rightarrow sk, pk \\ \qquad sk, pk, g, m \rightarrow nullifier, s, c \\ \qquad\qquad c = \text{hash}_2\left(g, pk, \text{hash}\left[m, pk\right], nullifier, \frac{g^s}{pk^c}, \frac{\text{hash}\left[m, pk\right]^s}{nullifier^c}\right) \\ \qquad\qquad \mathcal{A} \leftarrow nullifier, pk, pk_2, g, m \\ \qquad\qquad \mathcal{A} \rightarrow pk \end{array}\right) > 0.5 + negl$$

where $negl$ is the negligible function, and the adversary is effectively trying to tell if pk or arbitrary, valid pk_2 corresponds to the nullifier. We show that in the random oracle model, we can reduce the security of this to DDH.

Proof. Specifically, imagine an adversary Alice could in fact determine the correct public key with probability $0.5+\epsilon$, $\epsilon > negl$, using public signals. Then, imagine that Bob wanted to distinguish DDH; i.e. distinguish which of $(g, g^a, g^b, g^x), (g, g^a, g^b, g^y)$, for $x \neq ab, y = ab$, is the valid Diffie-Hellman tuple with epsilon advantage. Then Bob can randomly shuffle the tuples and designate the first one as g^x, then pass in $(nullifier = g^x, pk = g^a, pk_2 = r, g, m)$, where $pk_2 \neq pk$ is randomly generated. When Alice queries the hash oracle for hash$[m, pk]$, Bob can respond with g^a. When Alice queries pk, Bob can respond with g^b. Then, if Alice has an ϵ advantage, they can distinguish $g^a b$ in $1/2$ of the shuffles, so Bob has an $\epsilon/2$ advantage in solving decisional Diffie Hellman.

Chapter 4

Discussion

4.1 Performance

Our proof of concept for the nullifier uses a library called RustCrypto oriented towards constant time for all inputs, making timing-based side-channel attacks much harder [22]. We hope that ports to other languages prioritize this as well. Our code is OSS on GitHub *. It takes 0.35 seconds to compile, run, and verify without a ZK-SNARK.

The gas efficiency of our code is dominated by the Groth16 pairing check right now, but we expect that to be decreased when we switch to Nova.

4.2 Morality and Mitigations

We believe that as cryptographers, we have a moral imperative to release our work responsibly, and consider possible harms from releasing our research. We also deem it necessary to build in warnings and mitigations into such primitives before it is too late [31].

We hope that all uses of the nullifier will come with appropriate disclosures about the loss of anonymity upon ECDSA-breaking quantum supremacy.

*https://github.com/zk-nullifier-sig/zk-nullifier-sig

4.3 The Interactivity-Quantum Secrecy Tradeoff

Note that in the far future, once 256 qbit quantum computers can break ECDSA keypair security via breaking discrete log [32], most Ethereum keypairs will be broken, but funds will be safe. People can merely sign messages committing to new quantum-resistant keypairs (or just higher bit keypairs on similar algorithms), and the canonical chain can fork to make such keypairs valid. ZK-SNARKs become forgeable as their soundness guarantees are broken [36], for instance via a similar discrete log calculating toxic waste. However, they still guarantee, for statistically zero knowledge systems euch as Groth16, that secret data in past proofs still cannot ever be revealed. In the best case, the blockchain's guarantees should all continue to be upheld.

However, if people rely on any type of deterministic nullifier like our construction, their anonymity is immediately broken: someone can merely derive the secret keys for the whole anonymity set, calculate all the nullifiers, and see which ones match. This problem will exist for any deterministic nullifier algorithm on ECDSA, since revealing the secret key reveals the only source of "randomness" that guarantees anonymity in a deterministic protocol.

If people want to keep post-quantum secrecy of data, you have to give up at least one of our properties: the easiest one is probably non-interactivity. For example, for the zero knowledge airdrop, each account in the anonymity set publicly signs a commitment to a new semaphore id commitment (effectively address pk publishes hash[randomness | external nullifier | pk]) [17]. Then to claim, they reveal their external nullifier and ZK prove it came from one of the semaphore ids in the anonymity set. This considerably shrinks the anonymity set to everyone who has opted into a semaphore commitment prior to that account claiming, meaning the first claimer can potentially have anonymity set of size 1. The loss of noninteractivity also means some constructions such as the tornado cash improvement via our present nullifier construction is impossible (see use cases). However, since hashes (as far as we currently know) are still hard with quantum computers, it's unlikely that people will be able to ever de-anonymize you.

We hope that people will choose the appropriate algorithm for their chosen point on the interactivity-quantum secrecy tradeoff for their application, and hope that including this information helps folks make the right choice for themselves. Folks prioritizing shorter-term secrecy, like DAO voting or short-term confessions of the young, might prioritize this document's nullifier construction, but whistleblowers or journalists might want to consider the semaphore construction instead. We also note that since it will take around 2321 signal qubits (not accounting for noise) to solve elliptic curve cryptography in 256 bit prime fields [30], we expect it to take several decades for this to become feasible – estimates range from 2050 to 2100, to never due to theoretical noise limitations.

4.4 Future Work

We expect to release a blog post to explain the algorithm intuitively. Finally, we hope that folks will use our open source repository on GitHub (so far with a Rust proof of concept, Javascript to come) [†] to re-implement the scheme in a variety of formats and languages, especially ones compatible with different wallets. Although we considered publishing an IETF RFC for additional legitimacy, the existing RFC for the signature scheme alone (section 5 of [16]) seems sufficient for now.

[†]https://github.com/zk-nullifier-sig/zk-nullifier-sig/

Chapter 5

Conclusion

Having identified nullifiers as a key piece of cryptographic infrastructure limiting the use cases for ZK-SNARKs, in this work we explored the future of pseudonymous identity systems via nullifiers.

We demonstrated a construction of a new signature scheme with a public, deterministic nullifier component, and additional private signals that help verify the nullifier without the secret key. The nullifier both uniquely identifies the keypair and keeps the account identity secret. We proved uniqueness, secrecy, and existential unforgeability, and demonstrated a performant proof-of-concept.

We informally explained the quantum secrecy and interactivity tradeoff for nullifiers, and described possible mitigations for moral hazards introduced by this paper.

Bibliography

[1] Masayuki Abe, Jan Camenisch, Rafael Dowsley, and Maria Dubovitskaya. On the impossibility of structure-preserving deterministic primitives. *J. Cryptology*, 32(1):239–264, January 2019.

[2] Roman Storm Alexey Pertsev, Roman Semenov. Tornado.cash privacy solution. *tornado.cash*, 2019.

[3] Karim Baghery, Zaira Pindado, and Carla Ràfols. Simulation extractable versions of groth's zk-snark revisited. In *International Conference on Cryptology and Network Security*, pages 453–461. Springer, 2020.

[4] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE, May 2014.

[5] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing, 2001.

[6] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. https://toc.cryptobook.us/.

[7] Vitalik Buterin. Ethereum whitepaper, 2014.

[8] Vitalik Buterin. Some ways to use zk-snarks for privacy, Jun 2022.

[9] cabal.xyz team. cabal.xyz, 2022.

[10] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *Proc. Priv. Enhancing Technol.*, 2018(3):164–180, June 2018.

[11] Whitfield Diffie and Martin E Hellman. Multiuser cryptographic techniques. In *Proceedings of the June 7-10, 1976, national computer conference and exposition*, pages 109–112, 1976.

[12] A. Faz-Hernandez, R.S. Wahby, S. Scott, N. Sullivan, and C.A. Wood. Hashing to elliptic curves ietf rfc, Jun 2022.

[13] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

[14] Eu-Jin Goh and Stanisław Jarecki. A signature scheme as secure as the diffie-hellman problem. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 401–415. Springer, 2003.

[15] Sharon Goldberg, Dimitrios Papadopoulos, and Jan Včelák. Verifiable Random Functions (VRFs). Internet-Draft draft-goldbe-vrf-01, Internet Engineering Task Force, June 2017. Work in Progress.

[16] Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Včelák. Verifiable Random Functions (VRFs). Internet-Draft draft-irtf-cfrg-vrf-15, Internet Engineering Task Force, August 2022. Work in Progress.

[17] Kobi Gurkan, Koh Wei Jie, and Barry Whitehat. Community proposal: Semaphore: Zero-knowledge signaling on ethereum. *Accessed: Jul*, 1:2021, 2020.

[18] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. In *Lecture Notes in Computer Science*, Lecture notes in computer science, pages 147–176. Springer International Publishing, Cham, 2021.

[19] heyanon team. heyanon, 2022.

[20] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[21] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. *Cryptology ePrint Archive*, 2021.

[22] Nate Lawson. Side-channel attacks on cryptographic software. *IEEE Security & Privacy*, 7(6):65–68, 2009.

[23] Matt Lepinski, Silvio Micali, and Abhi Shelat. Fair-Zero knowledge. In *Theory of Cryptography*, Lecture notes in computer science, pages 245–263. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[24] Ueli M Maurer and Stefan Wolf. The relationship between breaking the diffie–hellman protocol and computing discrete logarithms. *SIAM Journal on Computing*, 28(5):1689–1721, 1999.

[25] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[26] Adhyyan Sekhsaria Nalin Bhardwaj, Aayush Gupta. Stealthdrop, 2022.

[27] Dimitrios Papadopoulos, Duane Wessels, Shumon Huque, Moni Naor, Jan Včelák, Leonid Reyzin, and Sharon Goldberg. Making nsec5 practical for dnssec. *Cryptology ePrint Archive*, 2017.

[28] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[29] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *International conference on the theory and application of cryptology and information security*, pages 552–565. Springer, 2001.

[30] Martin Roetteler, Michael Naehrig, Krysta M Svore, and Kristin Lauter. Quantum resource estimates for computing elliptic curve discrete logarithms. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 241–270. Springer, 2017.

[31] Phillip Rogaway. The moral character of cryptographic work. *Cryptology ePrint Archive*, 2015.

[32] Meryem Cherkaoui Semmouni, Abderrahmane Nitaj, and Mostafa Belkasmi. Bitcoin security with post quantum cryptography. In *International Conference on Networked Systems*, pages 281–288. Springer, 2019.

[33] Mehdi Tibouchi. A note on hashing to bn curves. *SCIS. IEICE*, 40, 2012.

[34] Sergei Volotikin. Software attacks on hardware wallets. *Black Hat USA*, 2018.

[35] Riad S Wahby, Dan Boneh, Christopher Jeffrey, and Joseph Poon. An airdrop that preserves recipient privacy. In *Financial Cryptography and Data Security*, Lecture notes in computer science, pages 444–463. Springer International Publishing, Cham, 2020.

[36] Yu Yu and Xiang Xie. Privacy-preserving computation in the post-quantum era. *National Science Review*, 8(9):nwab115, 2021.