Stateless 2PC Signatures for Internet-Scale Authentication and Authorization*

Michael Adjedj
Fireblocks
USA

Geoffroy Couteau†
Université Paris Cité
CNRS, IRIF
France

Arik Galansky
Fireblocks
USA

Nikolaos Makriyannis
Fireblocks
USA

Oren Yomtov
Fireblocks
USA

ABSTRACT

The industry is moving away from passwords for authentication and authorization, with hardware devices for storing long-term cryptographic keys emerging as the leading alternative. However, these devices often have limited displays and remain vulnerable to theft, malware, or tricking users into signing malicious payloads. Current systems provide little fallback security in such cases. Any solution must also meet strict requirements: compatibility with industry standards, scalability to handle high request volumes, and high availability.

Our approach virtualizes the authenticating/authorizing party via a two-party protocol with an informed helper. Keys are never assembled in one place, ensuring that the keys are protected even in the event of device compromise; all signatures are policy-checked; and the protocol is stateless, permitting helper *clones* to assume one another's work without coordination and thereby supporting *horizontal scalability*.

We formalize the required properties for such protocols and show how they are met by existing schemes (e.g., FROST for Schnorr, Boneh–Haitner–Lindell-Segev'25 for ECDSA). Motivated by the widespread use of ECDSA (FIDO2/Passkeys, blockchains), we introduce a new, optimized two-party ECDSA protocol that improves performance by about 4×–10× relative to the stateless BHLS'25 protocol at similar communication costs. At its core is a new variant of exponent-VRF, improving on earlier constructions and of independent interest. We validate our design with a proof-of-concept virtual authenticator for the FIDO2 Passkeys framework.

KEYWORDS

Passkeys, FIDO2, WebAuthn, Threshold Signatures, MPC

1 INTRODUCTION

The FIDO2 standard [14], commonly referred to as "passkeys," is a passwordless authentication protocol between a server (the *relying party*) and a user's authenticator, typically a hardware token or a smartphone. The authenticator generates a public–private key pair for each relying party and proves possession of the private key by signing server-issued challenges, often after local PIN or biometric verification. By eliminating reusable passwords, FIDO2 improves usability and offers strong protection against phishing and server-side credential breaches [15, 22].

However, FIDO2 still has critical security gaps. Private keys may be extracted if an authenticator is compromised—through device theft, malware, or weaknesses in key synchronization across devices. Moreover, authenticators have limited ability to verify the content or intent of what they sign, leaving users vulnerable to deception attacks in which they unknowingly authorize malicious actions. These shortcomings motivate the need for additional safeguards without undermining FIDO2's usability or interoperability.

In practice, this means that if an attacker compromises a device or deceives a user into approving a malicious request, they can generate valid signatures to authorize arbitrary actions. Such attacks have caused substantial real-world losses. For example, the Bybit exchange hack, in which attackers drained user funds by deceiving signing devices [32], and documented exploits against FIDO2/Passkeys that demonstrate similar vulnerabilities [9, 18, 23, 25, 33].

Mitigating the security concerns of FIDO2. The challenge of protecting a signing key while ensuring that every signed message complies with a security policy is not unique to FIDO2. Blockchain wallets face the same problem: every account is controlled by a private key that can authorize transfers of high-value assets, and attackers have repeatedly exploited stolen keys or tricked users into approving malicious transactions. In response, many commercial blockchain wallet providers have adopted the *MPC/virtualization paradigm*, in which the signing process is distributed between two or more parties (typically, the user and a dedicated service provider), so that no single party ever possesses the full key. At least one party is "informed," enforcing policy checks before authorizing any transaction, while the resulting MPC-generated signature remains indistinguishable from a regular one on the blockchain.

Our work adapts and extends this paradigm to the FIDO2 setting. This introduces new technical challenges: unlike blockchain wallets, where transaction rates are relatively low, FIDO2 must scale to millions of authentication events per second and maintain near-perfect availability, all while remaining compatible with existing infrastructure and standards (signatures).

Availability and scalability. For wide adoption, the informed party must remain available despite failures and be able to scale to millions of users. A natural approach is to deploy *clones*—identical

*This is the full version of a paper accepted to ASIA CCS 2026.
†Work done while the author was working as a consultant for Fireblocks.



This work is licensed under a Creative Commons Attribution 4.0 International License.

instances of the informed party running on multiple servers worldwide. Clones ensure fault tolerance by instantly taking over when another server fails, and improve scalability by distributing requests across servers. We refer to protocols that can securely support such cloning as supporting *horizontal scalability*.

Why naive cloning is insecure. Most MPC protocols for ECDSA and Schnorr are highly interactive and *stateful*, meaning that parties maintain state between communication rounds and that security crucially depends on correct state updates. This makes naive duplication insecure: replaying protocol messages to different clones can cause randomness to be reused, potentially leaking private keys. Even protocols that could, in principle, support horizontal scaling often lack formal security proofs in this setting. Moreover, cloning expands the attack surface—compromising any single clone along with one user can be sufficient to steal that user's credentials.

Our approach. We address these challenges by adapting the MPC/virtualization paradigm to FIDO2 authentication and authorization, designing protocols that enable secure horizontal scaling through *statelessness*. Intuitively, statelessness guarantees that security is preserved even if parties in the MPC crash (even mid-protocol) and are restored using only their long-term secrets.[1] Our solution is compatible with existing infrastructures, provides strong fallback guarantees even if a user's device is compromised or deceived, and achieves performance competitive with the most efficient stateful, i.e., non-scalable, protocols.

1.1 Our contributions

We propose a new authentication and authorization design based on the MPC/Virtualization paradigm, meeting the requirements outlined above, and provide a detailed security analysis. Our approach virtualizes the authenticating/authorizing party using a two-party protocol that includes a helper entity in the signing process. In summary:

(1) We show that any threshold protocol satisfying *unforgeability* and a set of additional properties we formalize can serve as a plug-in replacement, enabling full horizontal scalability when the authenticator/authorizer is split between two machines. Several state-of-the-art ECDSA [2, 7] and Schnorr [7, 19, 20] protocols meet these requirements fully (both parties are stateless) [7, 20] or partially (only one party is stateless) [2, 19].
(2) While some stateless Schnorr protocols achieve state-of-the-art efficiency, the same is not true for ECDSA: existing stateless protocols [7] are significantly slower than widely deployed threshold ECDSA schemes [12, 24]. Motivated by ECDSA's prevalence in FIDO2 and blockchain systems, we design a new, optimized two-party ECDSA protocol that satisfies statelessness while matching or exceeding the efficiency of the best in class stateful protocols. At its core is a *weakly-verifiable exponent VRF* (wv-eVRF), a relaxed analogue of the eVRF [7] that is also of independent interest.
(3) We validate our approach with a proof-of-concept that virtualizes the authenticator in the FIDO2/Passkeys framework.

Next, we expand on our contributions.

[1] From a security perspective, statelessness and horizontal scalability are equivalent.

1.1.1 *Statelessness via resettable security.*

Duplicating one of the MPC signatories without communication between clones introduces new attack vectors. We show that *resettable security*, which naturally captures the risks of duplication, suffices to guarantee statelessness. In our context, this means the protocol remains secure even if a party is reset to an earlier round—specifically, if an attacker resends a message from a previous round and the honest party proceeds as if it were receiving it for the first time. Similar notions of security have appeared in other contexts [3].

We also provide sufficient criteria for achieving resettable security (Section 3) and we show that several existing protocols from the literature satisfy this notion. Ultimately, our goal is for resettable-secure protocols to be composable with other components (e.g., as a virtual authenticator in the FIDO2 proposal) and to formalize the role of an *informed party*—one that only signs messages compliant with a pre-agreed policy. In Appendix B, we show that any resettable-secure protocol can be adapted to UC-realize an ideal policy-compliant signature functionality.

1.1.2 *Highly Efficient* wv-eVRF.

The protocol of [7] introduced the notion of an *exponent verifiable random function* (eVRF), a pseudorandom primitive that allows a sender to provably generate a group element A whose discrete logarithm is the output of a PRF on an input known to all parties. This primitive enables the design of two-round deterministic ECDSA signature protocols in a generic way. However, the two instantiations of the eVRF functionality presented in [7], one based on Paillier encryption and one based on the DDH assumption in the random oracle model, still incur substantial performance overhead. Specifically, the Paillier-based construction requires over 1 s of computation and 14 kB of communication, while the DDH-based version requires between 400 ms and 1 s of computation and about 2 kB of communication. For comparison, the entire protocol of [2] produces an ECDSA signature in roughly 50 ms with 2 kB of communication.[2]

We address this gap by:

(1) Introducing a *weakly verifiable* eVRF (wv-eVRF) that relaxes verifiability to a designated party, sufficient for two-party settings.
(2) Showing that our wv-eVRF realizes a modified eVRF functionality enabling two-round threshold ECDSA with a single eVRF call; protocols from [7] can thus be sped up by $\approx 2\times$.
(3) Designing a wv-eVRF with < 50 ms computation and 1.3 kB communication, usable as a drop-in replacement in [7, 20] or to add statelessness to [2, 19].

Integrating our optimized eVRF into [2] yields a fully scalable, two-round, resettably secure two-party ECDSA protocol with $\approx$ 100 ms total computation (both parties) and 3.3 kB communication at 128-bit security (3072-bit RSA modulus).

1.1.3 *Proof-of-Concept.*

We developed an open source proof-of-concept (PoC) for thresholdizing FIDO2 authentication that uses ECDSA signatures, which we call *passkeys_mpc*. Our open-source implementation is available at the referenced repository [27]. This PoC includes a modified version of a software authenticator with a

[2] See [2, Section 1.3] for a detailed discussion of these estimates.

cryptographic back-end that delegates all sensitive ECDSA operation (e.g., key generation and signature) to an MPC engine. It was designed to be compatible with existing FIDO2 relying parties and was validated using the Yubico Demo Server [1].

1.1.4 Additional technical contribution. As noted above, full horizontal scalability requires many servers sharing long-term cryptographic material. For security, however, duplicating the same keys across all servers is risky—compromising one server could expose all users. For efficiency, storing user-specific material on every server is wasteful at the scale of hundreds of millions or billions of users. To address this, Appendix A presents a simple stochastic analysis of availability under the following constraints:

(1) Each server belongs to specific clusters defined by geographic proximity or another relevant attribute. Servers can be partitioned into t distinct clusters.
(2) Each user is connected with at least k servers in each cluster.
(3) Each server may experience downtime with a not-too-small probability α, capturing events such as maintenance or overload.
(4) With a small probability ε, an entire cluster of servers may go down, accounting for rare catastrophic events (e.g., weather-related disasters affecting servers in close proximity) or sophisticated attacks targeting specific groups of servers.

The goal is to choose t and k to ensure each user can reach at least one server with probability $1 - p$, where p is very small. For example, if each server fails with probability $\alpha = 1/2$ and an entire cluster fails with probability $\varepsilon = 1/100$, then with $t = 4$ clusters and each user connected to $k = 6$ servers per cluster, the failure probability is at most $p = 10^{-6}$. See Appendix A for details.

1.2 Related Work

Several works in the literature that address the concept of statelessness in threshold signatures.

Stateless and deterministic Schnorr. A number of works implement stateless and deterministic Schnorr signing. For example, [30] use Bulletproofs over carefully chosen curves, [17] use garbled circuits, and [20] use pseudorandom correlation functions (PCFs).

Exponent VRFs and deterministic ECDSA. [7] introduce the notion of an *exponent verifiable random function* (eVRF) as a standalone primitive and use it to construct deterministic variants of several interactive protocols, including Schnorr and ECDSA. Notably, the deterministic protocols discussed earlier [17, 20, 30] can be viewed as specific instantiations or variants of the eVRF functionality.

Our approach. While full determinism (and the associated statelessness) implies resettable security, it is stronger than what is necessary for our application. In our setting, the right flavor of resettable security suffices to ensure security under duplication and horizontal scaling, without incurring the performance overhead of enforcing full determinism. Accordingly, our work focuses on achieving resettable security directly, and our proposed protocols do not aim to provide full determinism, allowing us to retain efficiency comparable to the fastest stateful protocols.

1.3 Paper Organization

In Section 2, we introduce preliminaries. In Section 3, we define signing protocols and the associated security notions. In Section 4, we present our new resettably secure ECDSA protocol. In Section 5, we introduce the notion of wv-eVRF and provide an instantiation based on Paillier encryption. Finally, in Section 6, we discuss our open-source proof of concept.

2 PRELIMINARIES

2.1 Notation

In this document, $\mathbb{Q}, \mathbb{Z}$ and $\mathbb{N}$ denote the set of rational, integer and natural numbers, respectively, and $\mathsf{PRIMES}^{>2}$ denotes the set of odd primes. Upper case bold letters $\boldsymbol{X}, \boldsymbol{S}, \ldots$ denote sets and lower case bold letters $\boldsymbol{u}, \boldsymbol{v}, \ldots$ denote random variables.

2.1.1 Groups and Fields. The *group-tuple* $(\mathbb{G}, g, q)$ will denote the group-generator-order tuple for ECDSA (technically, the description of the group is a deterministic function of the security parameter but to alleviate notation we do not explicitly write this). We write $\mathbb{Z}_q$ to specify that the field has q elements. For $N \in \mathbb{N}$, we write $\mathbb{Z}_N$ for the quotient group $\mathbb{Z}/N\mathbb{Z}$ and, $\mathbb{Z}_N^*$ and QR_N for the group of units and quadratic residues, respectively. For $\ell \in \mathbb{Z}$, we let $\pm\ell$ denote the interval of integers $\{-|(\ell-1)/2|, \ldots, 0, \ldots, |(\ell-1)/2|\}$ if ℓ is odd and $\{-|\ell/2|+1, \ldots, 0, \ldots, |\ell/2|\}$ otherwise.

2.1.2 Algorithms, Polynomials & Negligible Functions . Machines that fulfill a special function are referred to as *oracles* to distinguish them from other machines. We write $x \xleftarrow{\$} \mathbf{E}$ or $x \xleftarrow{\$} \boldsymbol{b}$ for sampling x uniformly from a set $\mathbf{E}$ or as a sample of $\boldsymbol{b}$ respectively, and $x \xleftarrow{\$} \mathsf{A}$ or $x \xleftarrow{\$}$ keygen for sampling x according to (probabilistic) algorithms A or keygen respectively—when the algorithm is deterministic or the operation correspond to (non-randomized) assignment, we simply write '$\leftarrow$' instead of '$\xleftarrow{\$}$'. We write $\mathrm{neg}(\cdot)$ for the set of negligible functions (functions that vanish faster than any inverse-polynomial).

2.1.3 Oracle-Aided Algorithms. Oracles are stateful interactive machines that fulfill a special function. An oracle-aided algorithm A with access to an oracle O can make queries to O and receive answers from O. Formally, A and O share a dedicated input-output tape for communicating queries and responses. Moreover, O may interact with multiple machines—say, A and B—where the query/answer pairs of one machine may depend on those of the other.

2.1.4 ECDSA Signatures. For a group tuple $(\mathbb{G}, g, q)$ let $\chi : \mathbb{G} \to \mathbb{Z}_q$ and $\mathsf{F} : \{0,1\}^* \to \mathbb{Z}_q$ denote the ECDSA conversion function (which is simply a deterministic function known to all) and the ECDSA message digest function which maps arbitrary strings to elements modulo q. ECDSA public-secret pairs have the form $(x, X) \in \mathbb{Z}_q \times \mathbb{G}$ such that $X = g^x$. ECDSA signatures for a message msg with digest $m = \mathsf{F}(\mathrm{msg})$ have the form $(r, \sigma) = (\chi(g^k), (m + rx)/k) \in \mathbb{Z}_q^2$ for $k \xleftarrow{\$} \mathbb{Z}_q^*$. Finally, (r, σ) is a valid signature for message digest m and public key X if $\chi((g^m \cdot X^r)^{1/s}) = r$.

2.2 Security Assumptions

Next we define the existential unforgeability of plain, non-threshold ECDSA and the knowledge of exponent assumption.

2.2.1 *Unforgeability of ECDSA.*

Experiment $\mathsf{Unf}_{\mathrm{ECDSA}}(\mathsf{A}, \ell, z)$

Let A denote an attacker initialized on security parameter ℓ and auxiliary input z. Let $(\mathbb{G}, g, q)$ denote the group tuple. The experiment is defined as follows:

(1) Sample $x \leftarrow \mathbb{Z}_q$ and returns $X = g^x$ to $\mathcal{A}$.
(2) Adaptively, A chooses $s(\ell)$ polynomialy-many messages digest $(m_i)_{i \leq s(\ell)}$. When m_i is chosen, A obtains a random ECDSA signature for m_i and X.

Output. A outputs $(m, (r, \sigma)) \in \{0,1\}^*$.

Experiment 1: Existential Unforgeability of plain ECDSA

Definition 2.1. Consider an execution of Experiment 1. We say that ECDSA is existentially unforgeable if for every adversary A it holds that

$$\Pr_{(m,(r,\sigma)) \xleftarrow{\$} \mathsf{Unf}_{\mathrm{ECDSA}}} \left[m \notin (m_i)_{i \leq s(\ell)} \wedge \mathcal{X}((g^m \cdot X^r)^{1/s}) = r \right] \leq \mathrm{neg}(\ell).$$

2.2.2 *Knowledge of exponent assumption.*

Definition 2.2. For an algorithm A, we write $\rho \in \{0,1\}^*$ and $z \in \{0,1\}^*$ to denote its random tape and auxiliary input respectively. Letting $(\mathbb{G}, g, q)$ denote a group tuple, we say that *KEA* holds true if for any algorithm A and auxiliary input z, there exists an algorithm $\mathcal{E}$ such that

$$\Pr_{\substack{g^\alpha \xleftarrow{\$} \mathbb{G} \\ \rho \xleftarrow{\$} \{0,1\}^*}} \left[\begin{cases} (B, C) \leftarrow \mathsf{A}(g^\alpha, z, \rho) \\ \beta \leftarrow \mathcal{E}(g^\alpha, z, \rho) \end{cases} \text{ s.t. } C = B^\alpha \wedge B \neq g^\beta \right] \leq \mathrm{neg}(\ell)$$

3 TWO-PARTY SIGNING PROTOCOLS

In this section, we define two-party signing protocols. We note that in this paper, signing protocols only deliver output to only one of the parties (namely, P_1). Additionally, we define following security notions for a signing protocol Σ.

(1) Standard unforgeability, where it is infeasible for the the attacker to generate a forgery in an execution of Σ.
(2) Unforgeability under reset attacks, where it is infeasible for the the attacker to generate a forgery even with the extra capability of being able to reset the honest party at a previous stage of the protocol. This is the target security notion that we wish to achieve for our protocols, as it automatically guarantees security when one of the parties is duplicated (with its entire state and random tape) across several, non-communicating machine, and a malicious user interacts with multiple clones.
(3) Semi-deterministic signatures, where, when fixing the randomness of the output-receiving party P_1, if P_1 outputs a valid signature, then this output is unique, no matter what the adversary does.
(4) Abort predictability, which, intuitively, states that the adversary's view of the protocol determines whether the signing protocol will result in a valid signature or not.

In this section, we show that any two-round, non-pathological protocol is resettable secure against the non-output-receiving party P_2, and that any two-round protocol satisfying Items 1, 3 and 4 is fully resettable secure (i.e., it also meets Item 2). Interestingly, although we do not provide a formal proof here, it follows that the two-round protocols in [2] for ECDSA and [19, 21] for Schnorr are resettable secure against P_2, whereas the eVRF-based Schnorr and ECDSA protocols from [6] and the Schnorr protocol from [21] are fully resettable secure against both parties, as they satisfy Items 1, 3 and 4.

3.1 Definitions and Security Notions

Definition 3.1. A two-party signing protocol, denoted by $\Sigma = (\mathsf{P}_1, \mathsf{P}_2)$, is an interactive process between two parties P_1 and P_2, modeled as oracle-aided PPTMs with access to some oracle O. The protocol consists of two distinct phases: key-generation and signing. In each phase, the parties exchange messages in a sequence of rounds, with only one party speaking in any given round. The messages exchanged in the key-generation phase are specified by an algorithm init, while those exchanged in the signing phase are specified by an algorithm sign. In more detail:

Key-Generation Phase:

(1) The algorithm init takes as input:
- the security parameter ℓ,
- the signatory identity $b \in \{1, 2\}$,
- local randomness ρ,
- the transcript so far τ,
- the return-values of queries to O,

and it outputs a message μ for the next round. Note that the queries to O are determined by the tuple (ρ, b, τ).
(2) At the end of key generation, if both parties are honest, party P_i outputs the pair $(\mathsf{V}, \mathsf{ss}_i)$, where V is the verification algorithm and ss_i is the local secret share.

Signing Phase:

(1) The algorithm sign takes as input:
- the security parameter ℓ,
- the signatory identity $b \in \{1, 2\}$,
- the pair $(\mathsf{V}, \mathsf{ss}_i)$,
- local randomness ρ,
- the transcript so far τ,
- the return-values of queries to O,

and it outputs a message μ for the next round. Note that the queries to O are determined by the tuple $(\rho, b, \mathsf{V}, \mathsf{ss}, \tau)$.
(2) At the end of the signing phase, if both parties are honest, party P_1 outputs a signature σ, while P_2 outputs nothing.

3.1.1 *Unforgeability under reset attacks.*

Experiment $\mathsf{ExUnf}_{\Sigma}(\mathsf{A}, \ell, z)$

A denotes an adversary with oracle access to O that takes input the security parameter ℓ and auxiliary parameter z

in an execution of protocol Σ. The experiment is defined as follows:

(1) A sends messages on behalf of P_i to the *honest party* P_{3-i} and we say that A *corrupts* P_i.
(2) The key-generation phase is ran once at the beginning of the execution. Let (V, ss_{3-i}) denote the honest party's output during key-generation.
(3) After the key-generation phase, $s(\ell)$ polynomially-many signing sessions are run between P_1 and P_2, without any restrictions on concurrency, where the messages for signing are adaptively chosen by A. Let $(m_i)_{i\le s(\ell)}$ denote the messages chosen by A.
(4) During any instance of the signing phase, the adversary A is allowed to reset ($r(\ell)$ polynomially many times) party P_{3-b} by sending a message on behalf of P_i corresponding to a round that has already transpired (the message does not have to be the same as the one originally sent). After the reset, the honest party continues the execution using the same inputs and random tape. However, if a corrupted P_1 re-sends the first message of the protocol, then P_2 is reinitialized and a new random tape is chosen, as if a new signing instance were initiated.
(5) A obtains the output of the honest party in every signing instance, including the ones obtained by resetting the honest party (so A potentially obtains many signatures for the same signing instance).

Output. A outputs a pair $(m, \sigma) \in \{0,1\}^*$.

Experiment 2: Existential Unforgeability under Reset Attacks

Definition 3.2. Let Σ denote a two-party signing protocol and consider an execution of Experiment 2. We say that Σ is unforgeable under reset attacks if for every adversary A it holds that

$$\Pr_{(m,\sigma)\xleftarrow{\$}\mathsf{ExUnf}_\Sigma(\mathsf{A},\ell,z)}\left[m \notin (m_i)_{i\le s(\ell)} \wedge \mathsf{V}(m,\sigma)=1\right] \le \mathrm{neg}(\ell).$$

Standard unforgeability is defined analogously, using the variant of Experiment 2 where the adversary does not have the reset capability described in Item 4 of the experiment.

3.1.2 Semi-deterministic Signatures.

Experiment $\mathsf{SemDet}_\Sigma(\mathsf{A}, \ell, z)$

A denotes an adversary with oracle access to O that takes input the security parameter ℓ and auxiliary parameter z in an execution of protocol Σ. The experiment is defined as follows:

(1) A sends messages on behalf of P_2 to the honest party P_1, i.e., A corrupts P_1.
(2) Same operations as Items 2 to 5 from Experiment 2. Write $\mathrm{out}_{i,j}$ for the output of P_1 in the i-th signing instance after the j-th reset.

Output. If there exists $i \in [s(\ell)]$ and $j, j' \in [r(\ell)]$ such that $\perp \neq \mathrm{out}_{i,j} \neq \mathrm{out}_{i,j'} \neq \perp$, output 1, else output 0.

Experiment 3: Semi-deterministic signatures experiment

Definition 3.3 (Semi-deterministic Signatures)**.** Let Σ denote a two-party signing protocol and consider an execution of Experiment 3. We say that Σ yields semi-deterministic signatures if for every adversary A it holds that

$$\Pr[\mathsf{SemDet}_\Sigma(\mathsf{A}, \ell, z) = 1] \le \mathrm{neg}(\ell).$$

3.1.3 Abort Predictability.

Definition 3.4 (Abort Predictability)**.** Let A denote an adversary corrupting P_2 in an execution of Σ and let out_i denote P_1's output in the i-th signing instance. We say that *abort-predictable* if the following holds for every A. There exists an algorithm S that on input

(1) the partial transcript of the protocol containing the key-generation and the first j signing instances,
(2) the query-answer pairs of A to the oracle O up to and including the j-th signing instance,

S returns $(b_1, \ldots, b_j) \in \{0,1\}^j$ such that

$$\Pr[\exists i \in [j] \text{ s.t. } (b_i = 0 \wedge \mathrm{out}_i \neq \perp) \vee (b_i = 1 \wedge \mathrm{out}_i = \perp)] \le \mathrm{neg}(\ell).$$

3.2 Two-Round Resettably-Secure Protocols

In this section, we give sufficient conditions for a two-round protocol to be secure under reset attacks.

Theorem 3.5. *Let Σ denote a two-round signing protocol and assume:*

(1) Σ is abort-predictable as per Definition 3.4.
(2) Σ yields semi-deterministic signatures as per Definition 3.3.
(3) Σ is unforgeable in the standard (non-resettable) sense as per Definition 3.2.

Then, it holds that Σ is unforgeable under reset attacks.

Proof. We first consider the case of corrupted P_1 which we discuss at a high-level as it is the easy case. Since Σ is a two-round protocol, the only reset a corrupted P_1 can perform is to reinitialize P_2 with a fresh random tape. As explained in Item 4, from P_2's perspective, this reset is equivalent to starting a new signing instance. Therefore, for two-round protocols, standard unforgeability against P_1 implies resettable security against P_1.

For the case of corrupted P_2, we show that any adversary that breaks resettable security of Σ can be turned into an adversary that break the standard unforgeability of the protocol. Let A denote an adversary that breaks the resettable security of Σ and consider the following attacker $\mathcal{B}$ in Experiment 2 without the reset capability. $\mathcal{B}$ relays messages between P_1 and A under the following caveats:

(1) When sending the second-round message for the first time in any signing instance (i.e., not during a reset attack), $\mathcal{B}$ first checks whether the message will cause an abort by using the predictor from Definition 3.4. If the predictor indicates an abort, $\mathcal{B}$ does not send the message and outputs $\perp$ as the honest party's result to A. Otherwise, $\mathcal{B}$ sends the message, receives the output out, records out as the valid output for that signing instance, and forwards it to A.

(2) If the adversary A resets P_1 by sending a different message and no valid output has been recorded, then $\mathcal{B}$ proceeds as described in Item 1. Else, if a valid output is recorded, $\mathcal{B}$ checks whether the message will cause an abort by using the predictor from Definition 3.4 and $\mathcal{B}$ forwards $\perp$ or out to A depending on the output of the predictor.

Thus, for protocols with abort predictors and semi-deterministic output, resettable security follows directly from standard unforgeability. □

4 SIMPLE TWO-PARTY ECDSA PROTOCOL

In this section, we introduce a simple 2-party ECDSA protocol that meet all security requirements of the previous section. The protocol builds upon two main ingredients:

- An eVRF functionality, that we instantiates in Section 5, and
- a "certified OLE" protocol. In our concrete efficiency estimates, we use the Paillier-based construction of [2] to instantiate it.

4.1 The eVRF Oracle

In this section, we introduce an ideal functionality for eVRF , to which both parties are given oracle access in our simple two-party ECDSA protocol. We note that our functionality differs significantly from the ideal eVRF functionality in [6]; we explain the rationale behind our variant, and describe a construction instantiating the functionality, in Section 5. We also note that our functionality is described in the single execution setting for simplicity, but it can be easily modify to handle parallel sessions by adding session identifiers.

Oracle eVRF

Parameters. Security parameter ℓ, group-tuple $(\mathbb{G}, g, q)$ and input domain $\mathbb{X} \subseteq \{0,1\}^*$.
Initialization. Upon receiving init from P_2, store init.
Evaluation. Upon receiving $(\mathsf{Eval}, \mathtt{val})$ with $\mathtt{val} \in \mathbb{X}$ from P_2, if init is stored, return x to P_2 such that
- If a tuple $(\mathsf{P}_2, \mathtt{val}, x')$ is stored, $x = x'$.
- Else, $x \xleftarrow{\$} \mathbb{Z}_q$ and store $(\mathsf{P}_2, \mathtt{val}, x)$.

Output. Upon receiving $(\mathtt{token}, \mathtt{val})$ from the ideal adversary, if a tuple $(\mathsf{P}_2, \mathtt{val}, x)$ is stored,
- If token = proceed, send $(\mathtt{val}, g^x)$ to P_1.
- Else, if token = abort, send $(\mathtt{val}, \perp)$ to P_1.

Oracle 1: The eVRF **functionality.**

4.2 The OLE Oracle

The oracle below implements a certified OLE functionality, that on input x from P_1 and (a, b, X) from P_2 sends $ax + b$ to P_1 provided that $X = g^x$. An efficient Paillier-based protocol instantiating this functionality was recently introduced in [2].[3]

[3]Strictly speaking, [2] implements a *leaky* variant of the functionality that allows some carefully-controlled leakage on x, and compensates for the leakage by introducing some modifications to the 2-party ECDSA protocol by perturbing x into $x' = x + q \cdot \mathsf{mask}$ to render the leakage statistically harmless while preserving the relation $g^{x'} = X$. For the sake of simplicity, we omit these technicalities in this work.

Oracle OLE

Parameters. Security parameter ℓ, group-tuple $(\mathbb{G}, g, q)$.
Inputs. P_1 holds $x \in \mathbb{Z}_q$ and P_2 holds $(a, b, X) \in \mathbb{Z}_q^2 \times \mathbb{G}$.
Output. If $X \neq g^x$, send $\perp$ to both, else:
(1) Send $ax + b \bmod q$ to P_1,
(2) Send nothing to P_2.

4.3 Protocol Description

Protocol 2PC-ECDSA

Parameters. Security parameter ℓ, group-tuple $(\mathbb{G}, g, q)$.
Inputs. P_i holds (X, x_i) for ECDSA pk $X = g^{x_1 + x_2}$.
Message to be signed. Message digest $m = \mathsf{F}(\mathsf{msg}) \in \mathbb{Z}_q^*$.
Oracles. OLE and eVRF with parameters ℓ, $(\mathbb{G}, g, q)$ and input domain $\mathbb{X} = \mathbb{G}^2 \times \mathbb{Z}_q$.
Setup. P_2 sends init to eVRF .
Operations.

Round 1. P_1:
Sample $k_1 \xleftarrow{\$} \mathbb{Z}_q$ and send $(R_1, C) \leftarrow (g^{k_1}, X_2^{k_1})$ to P_2.

Round 2.
(1) P_2: Abort if R_1 is the identy element or (R_1, X_2, C) is not a DH tuple, i.e., $R_1 = \mathsf{id}$ or $C \neq R_1^{x_2}$.
(2) Oracle: Call eVRF on input $(\mathsf{Eval}, (X, R_1, m))$ and receive an output $k_2 \in \mathbb{Z}_q$. Send $(\mathsf{proceed}, (X, R_1, m))$ to eVRF and write $((X', R_1', m'), R_2) \in \mathbb{X} \times \mathbb{G}$ for the output of P_1.
(3) P_2: Set $r \leftarrow \mathcal{X}(R_1^{k_2})$, $(a, b) \leftarrow (k_2^{-1} r, k_2^{-1}(m + r x_2))$.
(4) Oracle: Call OLE on inputs x_1 and (a, b, X_1) respectively. Let y denote P_1's output.

Output.
(1) P_1: Abort if $(X', R_1', m') \neq (X, R_1, m)$. Else, set $(r, \sigma) \leftarrow (\mathcal{X}(R_2^{k_1}), k_1^{-1} \cdot y)$. Output (r, s) if it is a valid signature for m.
(2) P_2 outputs nothing.

Protocol 1: Simple 2PC-ECDSA.

4.4 Security Claims

Theorem 4.1. *Assuming existential unforgeability of plain (non-distributed) ECDSA and KEA, it holds that* 2PC-ECDSA *is unforgeable in the standard (non-resettable) sense as per Definition 3.2.*

Proof. We prove security by showing that any adversary that breaks the unforgeability of 2PC-ECDSA can be transformed into an adversary that breaks the unforgeability of plain (non-distributed) ECDSA in Experiment 1. In particular, we show for any adversary

A, there exists a simulator S that can simulate the interaction of A and the honest party on a message digest m, for any number of signing instances.

Case 1. We begin with the case of a corrupted P_1. Let (r, σ) denote a valid signature for message digest m. Let R denote one of the group elements such that $\mathcal{X}(R) = r$, chosen at random.

When A hands over (R_1, C) as P_1's first-round message, the simulator extracts the discrete logarithm of R_1 guaranteed by Definition 2.2—if it fails to do so, S simply aborts. For the simulated messages for round 2, S sends:

(1) (X, R_1, m) and $R_2 = R^{k_1^{-1}} \in \mathbb{G}$ as the simulated output of the eVRF call.
(2) $y = k_1 \cdot \sigma \bmod q$ as the simulated output of the OLE call.

At the end of the simulation S outputs whatever A outputs. It is easy to see that the above simulation strategy can be extended to any number of signing instances without any restrictions on concurrency. Therefore, with access to a signature oracle for ECDSA, if A outputs a forgery with noticeable probability in Experiment 2, then S breaks the security of plain (non-distributed) ECDSA.

Case 2. Next we consider the case of a corrupted P_2. Let (r, σ) denote a valid signature for message digest m. Let R denote one of the group elements such that $\mathcal{X}(R) = r$, chosen at random. We define S as follows.

For P_1's simulated round-1 message, S samples $k_2 \xleftarrow{\$} \mathbb{Z}_q$, it sends $(R_1, C) \leftarrow (R^{k_2^{-1}}, R_1^{x_2})$ to A, and it updates the simulated eVRF to include the tuple $(\mathsf{P}_2, (X, R_1, m), k_2)$ as recorded. When A provides (X', R_1', m') and (a, b) for the eVRF and OLE call, abort if $(X', R_1', m') \neq (X, R_1, m)$ or $(a, b) \neq (k_2^{-1}r, k_2^{-1}(m + rx_2))$. Else, send (r, σ) to A.

We claim that, unless plain (non-distributed) ECDSA is insecure, then the aforementioned simulation process is perfect. Indeed, the only non-trivial item in the simulation is the test $(a, b) \neq (k_2^{-1}r, k_2^{-1}(m + rx_2))$, i.e., it may be the case in the protocol that $(a, b) \neq (k_2^{-1}r, k_2^{-1}(m + rx_2))$ and yet the protocol yields a valid signature in the end. In such a case, however, we argue that A actually managed to guess the secret share of the honest party, which in turn can be used to break the existential unforgeability of ECDSA. In more detail, we claim that for $a' = a - k_2^{-1}r$ and $b' = b - k_2^{-1}(m + rx_2) \bmod q$, if $(a', b') \neq (0, 0)$ and $(r, k_1^{-1}(ax_1 + b))$ is a valid signature, then $x_1 = -b'/a' \bmod q$. Indeed, since the signature is a deterministic function of (x_1, x_2, k_1, k_2), if $a' = 0$ then $b' = 0$ as well. On the other hand, if $a' \neq 0$, then a straightforward calculation yields that $x_1 = -b'/a' \bmod q$. In such a case, x_1 can be combined with x_2 to construct the secret key and generate forgeries in the simulation—which is ruled out, by assumption.

The above simulation strategy can be extended to any number of signing instances without any restrictions on concurrency. Therefore, like in the previous case, if A outputs a forgery with noticeable probability in Experiment 2, then S breaks the security of plain (non-distributed) ECDSA. □

Theorem 4.2. *It holds that* 2PC-ECDSA *yields semi-deterministic signatures as per Definition 3.3. Furthermore, assuming existential unforgeability of plain (non-distributed) ECDSA, it holds that* 2PC-ECDSA *is abort-predictable.*

Proof. We first argue that 2PC-ECDSA yields semi-deterministic signatures. First, note that P_1's output is a deterministic function of the tuple $(k_1, k_2, x = x_1 + x_2, m)$, where x is fixed in advance. In addition, notice that k_2 is determined by x, k_1, m thanks to the eVRF—indeed, on input (X, R_1, m) (which may be viewed as the encoding of (x, k_1, m)) the eVRF returns some value k_2. Therefore, when fixing m and P_1's random tape (which determines k_1), it follows that if a signing instance of 2PC-ECDSA yields a valid signature, then this signature is unique.

Next, we argue that 2PC-ECDSA is abort-predictable (assuming plain ECDSA is unforgeable). Define predictor S with access to the query-answer pairs of OLE and eVRF as follows: let k_2 and (a, b) denote the output of the eVRF and the input to the OLE, respectively, for a signing instance on message digest m, predictor S outputs 1 if $(a, b) = (k_2^{-1}r, k_2^{-1}(m + rx_2))$ and 0 otherwise. As in the proof of Theorem 4.1, if plain ECDSA is unforgeable, then the test $(a, b) = (k_2^{-1}r, k_2^{-1}(m + rx_2))$ determines whether P_1 outputs a valid signature. As such, 2PC-ECDSA is abort predictable. □

5 BOOSTING EFFICIENCY VIA A NEW EVRF

We present a novel exponent-VRF to generate deterministic pseudorandom $Z = g^z$ where P knows $z \in \pm 2^{n_x}$ and V knows Z as well as $\mathcal{E} = \mathsf{enc}_N(z)$. Our construction securely realizes the ideal eVRF functionality from Oracle 1, in a sense that we formalize below.

5.1 Definition

The definition uses two independent random oracles H and H′ that are treated differently in the analysis: $\mathsf{H} : \{0,1\}^\ell \to \mathbb{Z}_{N^2}^*$ is used to define the pseudorandom function underlying the eVRF, while H′ is used in zero-knowledge proofs for the Fiat-Shamir transformation. In the definitions below, we assume w.l.o.g. that 1^ℓ is included in vk and sk.

Definition 5.1. Let $\mathbb{G}$ be an ensemble of cyclic groups of order $q \approx 2^{2\ell}$ with a generator g. A *weakly-verifiable exponent-VRF* (wv-eVRF) with domain $\{0,1\}^\ell$, range $\mathbb{G}$, and function-families $\mathcal{H}, \mathcal{H}'$, is a triple of oracle PPT algorithms (KeyGen, Eval, Verify) such that for every integer ℓ and hash functions $\mathsf{H} \in \mathcal{H}_\ell$, $\mathsf{H}' \in \mathcal{H}'$:

- $\mathsf{KeyGen}^{\mathsf{H}}(1^\ell)$: returns a pair $(\mathsf{vk}, \mathsf{sk})$, where $\mathsf{vk} = (\mathsf{vk}_\mathsf{P}, \mathsf{vk}_\mathsf{V})$ is a verification key that can be syntactically split into a prover part vk_P and a verifier part vk_V, and sk is a secret key. We let $\mathsf{ProKG}^{\mathsf{H}}(1^\ell)$ denote KeyGen restricted to $(\mathsf{vk}_\mathsf{P}, \mathsf{sk})$ and $\mathsf{VerKG}^{\mathsf{H}}(1^\ell)$ denote KeyGen restricted to vk_V.
- $\mathsf{Eval}^{\mathsf{H},\mathsf{H}'}(\mathsf{sk}, \mathsf{vk}_\mathsf{V}, \mathtt{val})$: returns a triple $(x, X, \pi) = (\mathsf{Eval}_1^{\mathsf{H}}(\mathsf{sk}, \mathtt{val}), \mathsf{Eval}_2^{\mathsf{H}}(\mathsf{sk}, \mathtt{val}), \mathsf{Prove}^{\mathsf{H},\mathsf{H}'}(\mathsf{sk}, \mathsf{vk}_\mathsf{V}, \mathtt{val}))$.
- $\mathsf{Verify}^{\mathsf{H},\mathsf{H}'}(\mathsf{vk}, \mathtt{val}, X, \pi)$: returns a bit b.

Given a wv-eVRF (KeyGen, Eval, Verify), we let Π_{eVRF} denote the following two-party protocol between a prover P and a verifier V:

- P computes $(\mathsf{vk}_\mathsf{P}, \mathsf{sk}) \xleftarrow{\$} \mathsf{ProKG}(1^\ell)$ and sends vk_P. Simultaneously, V computes and sends $\mathsf{vk}_\mathsf{V} \xleftarrow{\$} \mathsf{VerKG}(1^\ell)$. Let $\mathsf{vk} = (\mathsf{vk}_\mathsf{P}, \mathsf{vk}_\mathsf{V})$.
- On common input val, if $\mathsf{Eval}^{\mathsf{H},\mathsf{H}'}(\mathsf{sk}, \mathsf{vk}_\mathsf{V}, \mathtt{val})$ does not return $\perp$, P computes $(x, X, \pi) \xleftarrow{\$} \mathsf{Eval}^{\mathsf{H},\mathsf{H}'}(\mathsf{sk}, \mathsf{vk}_\mathsf{V}, \mathtt{val})$ and sends (X, π) to the verifier.

- The verifier outputs X iff $\mathsf{Verify}^{\mathsf{H},\mathsf{H}'}(\mathsf{vk}, \mathtt{val}, X, \pi) = 1$.

A wv-eVRF is *secure* if the protocol Π_{eVRF} defined above securely realizes the eVRF functionality from Oracle 1 against an adversary maliciously corrupting at most one party.

5.2 Differences with BHL24

We compare our notion with that of [6], the full version of the Eurocrypt'25 paper, that contains significantly revised and updated definition compared to the conference version. Our definition of eVRF differs from the original definition from [6] on several important aspects. To emphasize the difference with the original definition and avoid confusion, we use a different name and call *weakly-verifiable* eVRF (wv-eVRF) our variant. Our notion is not directly comparable to the original eVRF notion: it weakens it on some aspects, but strengthen it on others. Below, we comment on the main differences and explain their motivation.

On the ideal functionality. First, the eVRF ideal functionality on Oracle 1 differs significantly from the ideal functionality from [6] (Definition 4 in [6]). In particular, our functionality is significantly simpler. This stems from two main reasons:

(1) First, we specialize the functionality to the two-party setting, which suffices for our applications, while [6] captures the use of eVRF in a multiparty setting.
(2) Second, and more importantly, our eVRF functionality returns a random output x to P_2 and g^x to P_1 even when P_2 is corrupted. In contrast, in [6], the functionality has a very different behavior when P_2 is corrupted:
    - During initialization, P_2 sends a deterministic polytime-computable function f together with init.
    - During evaluation on input val, when P_2 is corrupted, the functionality sets $x = f(\mathtt{val})$ and $X = g^x$.

The reason for allowing a corrupted P_2 to define a function f of its choice in [6] is that the point x is constructed in the real-world as a PRF evaluation on val with a secret key *known to* P_2—hence, in particular, it does not look random to P_2. This allows their functionality to be instantiable using any PRF (together with a suitable zero-knowledge proof of correct evaluation in the exponent).
We adopt a different strategy. We restrict our attention to specific PRFs *in the random oracle model*. Then, by letting the simulator get programmable access to the oracle, we simulate against a corrupted P_2 as follows: the simulator extracts the secret PRF key from P_2. Whenever P_2 queries the random oracle on a value val, the simulator first *queries the* eVRF *functionality* on input val in the ideal world on behalf of P_2, obtains an output x, "reverse engineers" the corresponding random oracle output (using the secret key of the PRF), and programs the random oracle accordingly.
Admittedly, this strategy is less general than the approach advocated in [6]. However, it nonetheless captures both our construction and all constructions in [6], with the minor caveat that the analysis against a corrupted P_2 now explicitly uses the underlying random oracle instead of invoking the PRF security in a black-box way. In addition, and beyond simplifying the functionality, it comes with a major benefit: this modification is the key that enables our simple 2-party ECDSA protocol to use a *single invocation* of the eVRF functionality. In contrast, all 2-party signing protocols in [6] require two invocations of the eVRF , one in each direction. To see this, let us outline the two "modes" in which the eVRFs can be employed within a 2-party ECDSA protocol.

- In [6], both P_1 and P_2 will simply compute their nonce by calling (two different instances of) the eVRF on (X, m). This ensures that the nonces are fixed ahead of time, and it lets the simulator extract them in the analysis by computing the corrupted party's nonce as $f(X, m)$ using the function f extracted during the initialization.
- In contrast, in our protocol, P_1 picks a random nonce and sends R_1 together with a proof of knowledge of the exponent. Then, P_2 sets their own nonce k_2 using an eVRF call on input (X, m, R_1). Observe that it is crucial to let P_2's nonce depend on R_1, as otherwise a malicious P_1 could run two signing instances with the same (X, m) but different R_1 and force P_2 to use the same R_2, which would immediately compromise unforgeability.

In the second mode, a key difference is that to simulate against a corrupted P_2, the simulator must set $R_1 = R^{k_2^{-1}}$ (where R comes from the signing oracle), where k_2 is the output of the eVRF *on input* (X, m, R_1), causing an apparent circularity issue. And indeed, using the ideal functionality from [6], simulation is impossible in this mode, as k_2 is fixed ahead-of-time as $f(X, m, R_1)$! In contrast, our modified eVRF functionality allows simulating by first choosing k_2 randomly, setting $R_1 = R^{k_2^{-1}}$, and then *programming the* eVRF *output* to be k_2.
We note that in spite of the significant differences between the functionalities, it is not too hard to show that both the Paillier-based and the DDH-based eVRF protocols from [6] actually instantiate our functionality in the ROM. Therefore, the 2-party protocols in [6] can alternatively use our "second mode" and replace the eVRF invocation in the first flow with a (considerably cheaper) proof of knowledge of the nonce.

Weak verifiability. Our eVRF definition allows the verifier to sample part of the verification key vk themselves. Since our construction instantiates the eVRF functionality when *at most one party* is corrupted, this implies in particular that external observers cannot be convinced of P_2's honest behavior unless they already trust that the verifier sampled vk_V honestly. Indeed, it is not too hard to show that our protocols satisfy the original eVRF definitions from [6] if vk_V is sampled by a trusted setup. However, when letting the verifier sample it, as we do, this prevents the eVRF from providing security guarantees to external observers, and makes our notion less appealing in a multiparty setting.
In turn, relaxing verifiability allows us to rely on much more efficient zero-knowledge proofs compared to [6]: we rely on approximate range proofs over groups of unknown order (combined with a simple observation, that seems to be new to our work, that approximate range proofs, as opposed to exact range proofs, suffice for security by adding a simple check) and let the verifier sample the modulus for the group of unknown order.

Simulation security. Eventually, we note that we directly provide a simulation-style security definition for our eVRF . In contrast, [6] provides a game-based notion, and proves that this notion together

with a suitable proof of knowledge of the eVRF key suffices to instantiate their functionality.

We observe that a similar result can be shown for our construction (using a suitable adaptation of the game-based security notions). However, adding this intermediate layer of abstraction would come at a cost: indeed, our concrete wv-eVRF construction directly instantiates our eVRF functionality *without* requiring the prover to add a proof of knowledge of the eVRF key—in our construction, the prover only proves that the eVRF key is well-formed, but the proof does not provide straight-line extraction of the key, which would have been much more expensive. This optimization stems from a specific feature of the Paillier cryptosystem: in the underlying Paillier-based PRF (that maps $\mathtt{val}$ to a random ciphertext and decrypt it), one can "program" the PRF outputs without knowing the Paillier secret key, by choosing the output, encrypting it, and programming the random oracle to return this ciphertext.

5.3 Construction

We represent on Protocol 2 our weakly-verifiable eVRF . It relies on multiple zero-knowledge proofs:

- The proofs $\Pi_{\mathsf{mod}}, \Pi_{\mathsf{fac}}$, and Π_{df} from [2]. Informally, Π_{mod} guarantees that N is a product of two primes and $\gcd(N, \phi(N)) = 1$, Π_{fac} guarantees that N does not have factors smaller than $\sqrt{N}/2^{\ell}$, and Π_{df} guarantees that $u = v^{\lambda} \bmod \hat{M}$ with $\lambda \leq 2^{2\ell+\nu}$.
- The proof Π_{dlenc} guarantees that $\mathcal{E}$ encrypts a not-too-large plaintext $x \leq 2^{n_x+\nu+1}$ such that $X = g^x$.

We recall all necessary preliminaries on $\Pi_{\mathsf{mod}}, \Pi_{\mathsf{fac}}$, and Π_{df} in Appendix C, and introduce Π_{dlenc} in Section 5.4.

Protocol Our wv-eVRF Construction

Common Input. Security parameter ℓ and group-description-tuple $(\mathbb{G}, g, h, f, q)$. Parameters $n_x = 2\ell$, $n_\lambda = 2\ell + \nu$. Hash functions $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}^*_{N^2}$ and $\mathsf{H}' : \{0,1\}^* \to \{0,1\}^{\ell}$.

- $\mathsf{ProKG}^{\mathsf{H}}(1^{\ell})$:
  - Sample $(N, n_{\mathrm{rsa}}; p_1, p_2) \xleftarrow{\$} \mathsf{SampleRSA}(1^{\ell})$ such that $(p_1, p_2) = (3, 7) \bmod 8$.
  - Generate Paillier-Blum well-formedness proof $\pi_{\mathsf{mod}} \xleftarrow{\$} \Pi^{\mathrm{FS}}_{\mathsf{mod}}(\mathrm{aux}, N; p_1, p_2)$.
  - Generate small factors proof $\pi_{\mathsf{fac}} \xleftarrow{\$} \Pi^{\mathrm{FS}}_{\mathsf{fac}}(\mathrm{aux}, N; p_1, p_2)$
  - Sample $\rho_0 \xleftarrow{\$} \mathbb{Z}^*_N$ and set $\rho = \rho_0^{2N} \bmod N^2$.
  - Output $\mathsf{vk_P} = (N, \rho, \rho_0, \pi_{\mathsf{mod}}, \pi_{\mathsf{fac}})$ and $\mathsf{sk} = \phi(N)$.
- $\mathsf{VerKG}^{\mathsf{H}}(1^{\ell})$:
  - Sample $(\hat{M}, n_{\mathrm{rsa}}, v, u; \hat{p}, \hat{q}, \lambda) \xleftarrow{\$} \mathsf{SampleDF}(1^{\ell})$.
  - Send $\pi_{\mathsf{df}} \xleftarrow{\$} \Pi^{\mathrm{FS}}_{\mathsf{df}}(\mathrm{aux}, \hat{M}, v, u; \lambda)$ to V.
  - Output $\mathsf{vk_V} = (\hat{M}, v, u, \pi_{\mathsf{df}})$.
- $\mathsf{Eval}^{\mathsf{H},\mathsf{H}'}(\mathsf{sk}, \mathsf{vk_V}, \mathtt{val})$
  1. Parse $\mathsf{sk} = \phi(N)$ and $\mathsf{vk_V} = (\hat{M}, v, u, \pi_{\mathsf{df}})$.
  2. Verify $\hat{M} \geq 2^{n_{\mathrm{rsa}}}$, $v, u \in \mathbb{Z}^*_{\hat{M}}$ and $\mathsf{vrfy}_{\Pi^{\mathrm{FS}}_{\mathsf{df}}}(\mathrm{aux}, \hat{M}, v, u, \pi_{\mathsf{df}}) = 1$. If verification fails, output $\perp$ and terminate.
     ▷ This step needs only to be executed once.
  3. Compute $(\mathsf{tmp}, r) \in \mathbb{Z}_N \times \mathbb{Z}^*_N$ such that $\mathsf{enc}_N(\mathsf{tmp}, r) = \mathcal{E}' = \mathsf{H}(\mathtt{val})$.
  4. Set $x \leftarrow [\mathsf{tmp}]_q$ and $X \leftarrow g^x$.
  5. Set $\sigma \leftarrow (\mathsf{tmp} - x)/q$, and $\mathcal{E} \leftarrow \mathcal{E}' \cdot (1 - \sigma N) \bmod N^2$.
  6. Generate $\psi_{\mathsf{dlenc}} \xleftarrow{\$} \Pi^{\mathrm{FS}}_{\mathsf{dlenc}}(ssid, \mathsf{P}, \mathbb{G}, g, q, X, \mathcal{E}, N, \rho, n_x, n_\lambda, \hat{M}, v, u; x, \beta)$
  7. Set $\pi = (\sigma, \psi_{\mathsf{dlenc}})$ and output (x, X, π).
- $\mathsf{Verify}^{\mathsf{H},\mathsf{H}'}(\mathsf{vk}, \mathtt{val}, X, \pi)$
  1. Parse $\mathsf{vk} = (\mathsf{vk_P}, \mathsf{vk_V})$, $\mathsf{vk_P} = (N, \rho, \rho_0, \pi_{\mathsf{mod}}, \pi_{\mathsf{fac}})$, $\mathsf{vk_V} = (\hat{M}, v, u, \pi_{\mathsf{df}})$, and $\pi = (\sigma, \psi_{\mathsf{dlenc}})$.
  2. Verify $N \geq 2^{n_{\mathrm{rsa}}}$, $\rho_0 \in \mathbb{Z}^*_N$, $\rho = \rho_0^{2N} \bmod N^2$, $\mathsf{vrfy}_{\Pi^{\mathrm{FS}}_{\mathsf{mod}}}(\mathrm{aux}, N, \pi_{\mathsf{mod}}) = 1$, and $\mathsf{vrfy}_{\Pi^{\mathrm{FS}}_{\mathsf{fac}}}(\mathrm{aux}, N, \pi_{\mathsf{fac}}) = 1$
     ▷ This step needs only to be executed once.
  3. Set $\mathcal{E} \leftarrow \mathsf{H}(\mathtt{val}) \cdot (1 - q\sigma N) \bmod N^2$.
  4. Check that $q\sigma \in \pm N - \sqrt{N}$
  5. Check that $\mathsf{vrfy}_{\Pi^{\mathrm{FS}}_{\mathsf{dlenc}}}(ssid, \mathsf{P}, \mathbb{G}, g, q, X, \mathcal{E}, N, \rho, n_x, n_\lambda, \hat{M}, v, u, \psi_{\mathsf{dlenc}}) = 1$.
  6. Output 1 if and only if all checks passed.

Protocol 2: Our wv-eVRF **construction**

In Appendix D, we formally prove that the protocol Π_{eVRF} obtained from Protocol 2 securely realizes the ideal eVRF functionality from Oracle 1 against a malicious adversary corrupting at most one of the parties.

5.4 Encrypted discrete log proof

The proof below is a variant of the Π_{dlenc} proof from [2], tailored to our setting.

Protocol Encrypted Discrete Log

Common Input. Group-tuple $(\mathbb{G}, g, q)$ and $X = g^x$, modulus N and $\rho_0 \in \mathbb{Z}^*_N$, $\rho = \rho_0^{2N} \bmod N^2$, parameters $(\hat{M}, u, v)$ and $\mathcal{E} \in \mathbb{Z}^*_{N^2}$ such that $\mathcal{E} = \mathsf{enc}_N(x; r^N) \bmod N^2$, $x \in \pm 2^{n_x}$ and $r \in \mathbb{Z}_N$.

Auxiliary Input. $\widetilde{P} = u^x v^{\mu} \bmod \hat{M}$ for $\mu \xleftarrow{\$} \pm 2^{n_\mu}$ where $n_\mu = 2\ell + n_\lambda + \nu$.

Operations. (Recall $\varepsilon = \ell + \nu$).

1. P samples $(\alpha, \lambda', \mu') \xleftarrow{\$} (\pm 2^{n_x+\varepsilon}) \times (\pm 2^{2\ell}) \times (\pm 2^{n_\mu+\varepsilon})$ and sends $\widetilde{P}$ and
$$\begin{cases} A = g^{\alpha} \in \mathbb{G} \\ \widetilde{B} = u^{\alpha} v^{\mu'} \bmod \hat{M} \\ \mathcal{D} = \mathsf{enc}_N(\alpha; \rho^{\lambda'}) \bmod N^2. \end{cases}$$
2. V sends $e \xleftarrow{\$} \pm 2^{\ell}$

(3) P returns (z_1, z_2, z_3) such that
$$\begin{cases} z_1 = \alpha + e \cdot x \\ z_2 = r^e \cdot \rho_0^{2\lambda'} \bmod N \\ z_3 = \mu' + e \cdot \mu \end{cases}$$
Verifier's Output. Accept if $z_1 \in \pm 2^{n_x+\varepsilon}$ and
$$\begin{cases} g^{z_1} = A \cdot X^e \in \mathbb{G} \\ u^{z_1} v^{z_3} = \widetilde{B} \cdot \widetilde{P}^e \bmod \hat{M} \in \mathbb{Z}^*_{\hat{M}} \\ (1+N)^{z_1} \cdot z_2^N = \mathcal{D} \cdot \mathcal{E}^e \bmod N^2 \in \mathbb{Z}^*_{N^2} \end{cases}$$

Protocol 3: Encrypted Discrete Log Proof Π_{dlenc}.

Claim 5.2 (Zero-Knowledge). *Let Π'_{dlenc} denote the protocol Π_{dlenc} with the following modifications:*

- *In step (1),* P *samples $\eta \xleftarrow{\$} \mathbb{Z}^*_N$ and sets $\mathcal{D} = (1+N)^\alpha \cdot \eta^N \bmod N^2$.*
- *In step (3),* P *sets $z_2 = r^e \cdot \eta \bmod N$.*

Then,

1. *The protocol Π'_{dlenc} is $2^{-\nu+1}$-HVZK*
2. *The advantage of any PPT (possibly malicious) verifier V^* in distinguishing between a transcript of Π_{dlenc} and of Π'_{dlenc} is at most $\mathsf{Adv}^{\mathsf{sei}}_{\mathsf{V}^*}(1^\ell)$.*

The first part of the claim is standard; a near-identical proof can be found e.g. in Proposition 5.5 of [2]. The second part of the claim follows via a straightforward reduction to SEI that receives either $\rho_0^{2\lambda'}$ for a random 2ℓ-bit exponent λ' or a random element η of $\mathbb{Z}^*_N$.

Remark 5.3. Using Π_{dlenc} instead of Π'_{dlenc} sacrifices statistical HVZK, but significantly improves computational efficiency for P, since it replaces one full-sized exponentiation by N with a much cheaper exponentiation by the 2ℓ-bit exponent λ'.

Claim 5.4 (Special Soundness). *Let* $\mathsf{trans}, \mathsf{trans}'$ *denote two accepting transcripts of Protocol 3 sharing the same $(A, \tilde{B}, \mathcal{D})$ and with distinct $e \neq e'$. There exists an extractor* E *such that at least one of the following holds true.*

1. *$(\alpha, \beta, \gamma) \leftarrow \mathsf{E}(\mathsf{trans}, \mathsf{trans}')$ such that $\gamma \not| \alpha$ and $u^\alpha v^\beta = \tilde{P}^\gamma \bmod \hat{M}$.*
2. *$k \leftarrow \mathsf{E}(\mathsf{trans}, \mathsf{trans}')$ such that $k < 2^\ell$ and $\gcd(k, N) \neq 1$.*
3. *$(x, r) \leftarrow \mathsf{E}(\mathsf{trans}, \mathsf{trans}')$ such that*
   (a) *$(x, r) \in \pm 2^{n_x+\nu+1} \times \mathbb{Z}^*_N$,*
   (b) *$X = g^x \in \mathbb{G}$ and $\mathcal{E} = (1+N)^x r^N \bmod N^2$.*

In particular, this implies that assuming the hardness of strong RSA and N does not admit small factors, it necessarily holds that the extractor produces a pair (x, r) such that $\mathcal{E}$ is a Paillier encryption of x with randomness r, and $X = g^x$.

Proof. Let $(\Delta_e, \Delta_1, \Delta_2, \Delta_3) \leftarrow (e'-e, z'_1-z_1, z'_2/z_2 \bmod N, z'_3-z_3)$. By the verification equations,

- $u^{\Delta_1} v^{\Delta_3} = \tilde{P}^{\Delta_e} \bmod \hat{M}$,
- $g^{\Delta_1} = X^{\Delta_e} \in \mathbb{G}$ and $(1+N)^{\Delta_1} \cdot \Delta_2^N = \mathcal{E}^{\Delta_e} \bmod N^2$.

Table 1: Count of the dominant operations in eVRF and in Π_{dlenc}, given as a number of decryption operations (dec), full exponentiations (FE) and as the total length of exponents involved in short exponentiations (SE)

	$\mathbb{Z}^*_{N^2}$ dec	$\mathbb{Z}^*_{N^2}$ SE	$\mathbb{Z}^*_{N^2}$ FE	$\mathbb{Z}_N$ SE
eVRF (P)	1	0	0	0
Π_{dlenc} (P)	0	2ℓ	0	$17\ell + 6\nu$
Π_{dlenc} (V)	0	3ℓ	1	$9\ell + 4\nu$
Total (P)	1	2ℓ	0	$17\ell + 6\nu$
Total (V)	0	1	1	$9\ell + 4\nu$

If $\Delta_e \not| \Delta_1$, output $(\Delta_1, \Delta_3, \Delta_e)$ thus satisfying Item 1. Else, if $\gcd(\Delta_e, N) \neq 1$, output Δ_e as per Item 2. Else, output (x, r) as follows. Set $x \leftarrow \Delta_1/\Delta_e$. This immediately implies that $x \in \pm 2^{n_x+\varepsilon+1}$ and $X = g^x$ since Δ_e is coprime to q. The last equation becomes $\Delta_2^N = (\mathcal{E} \cdot (1+N)^{-x})^{\Delta_e} \bmod N^2$. We use the following standard fact (Fact 5.11 in [2])

Fact 5.5. *Let $M, k, \ell \in \mathbb{N}$ such that $\gcd(k, \ell) = 1$ and $k, \ell \leq M$. For $C, x \in \mathbb{Z}^*_M$ such that $C^k = x^\ell \bmod M$, there exists efficiently computable (in $|M|$) $y \in \mathbb{Z}^*_M$ such that $C = y^\ell \bmod M$.*

Using the above fact and since $\gcd(\Delta_e, N) = 1$, we compute $r \in \mathbb{Z}^*_{N^2}$ such that $r^N = \mathcal{E} \cdot (1+N)^{-x}$, hence $\mathcal{E} = (1+N)^x \cdot r^N \bmod N^2$, and the conditions of Item 3 are satisfied. □

5.5 Concrete Efficiency

Complexity estimates. We provide on Table 1 a count of the most costly operations used in our new eVRF . In the table, SE stems for "short exponentiation" (exponentiation with an exponent much shorter than n_{rsa}) and FE for "full exponentiation" (exponentiation by N). For short exponentiations over $\mathbb{Z}_{N^2}$ or $\mathbb{Z}_N$, we report the total length of the exponents, as the computational cost scales linearly with the exponent length. For full exponentiations, we report the number of full exponentiations.

Comparison with the state of the art. On Table 2, we compare our new eVRF to existing constructions from the literature:

- The DDH-eVRF and the Paillier-eVRF from [6]
- The construction from [20], that can be viewed as a *designated-verifier* eVRF (as the work of [20] predates [6], they do not employ this terminology, but their construction can replace any eVRF in a 2-party setting).

We do not consider the construction of [30], that appears to be less efficient than more recent works. Regarding runtime estimates, we use the estimates from [8], but stress that those come from preliminary implementations (for Paillier-eVRF) or rough estimates (for DDH-eVRF). Consequently, we provide a rather large confidence interval for DDH-eVRF (current runtimes estimates are on the higher end of our interval, but could significantly improve with future optimized implementations). For [20], we use the runtimes from their work, that were obtained on a single thread of a laptop (with a Ryzen 7 5800H processor). We note that [20] did not provide a concrete protocol for generating the verification key, but assumed

Table 2: Comparison of various eVRF constructions

Scheme	n_{rsa}	Runtime	Proof size
DDH-eVRF [8]	✗	200ms~500ms	1kB
Paillier-eVRF [8]	2048	160ms	14kB
PCF-eVRF [20]	3072	186ms	514B
Ours	2048	16ms	0.9kB
Ours	3072	49ms	1.3kB

a trusted setup; finding an efficient protocol to instantiate their trusted setup remains to be done (it is likely that the techniques used in our work could be adapted to design such a protocol).
We implement our wv-eVRF and report the runtime we encountered (16ms for 2048-bit modulus and 50ms for 3072). We consider both $n_{\text{rsa}} = 2048$ and $n_{\text{rsa}} = 3072$ to facilitate the comparison with [6] (whose numbers assume $n_{\text{rsa}} = 2048$) and [20] (whose numbers assume $n_{\text{rsa}} = 3072$). We set the security parameter ℓ to 128 and the statistical security parameter ν to 64. Our benchmarks were obtained on a single thread of a laptop with an Intel(R) Core(TM) i7-1365U CPU. When integrating our new eVRF within the recent 2-party 2-round ECDSA protocol from [2], we obtain an end-to-end runtime estimate of 100ms for a 3072-bit modulus at the 128-bit security level, and an end-to-end communication around 3.3kB. While the real runtime of a full-fledged implementation can vary compared to our numbers, they should provide close estimates of the real runtime.

6 "THRESHOLD PASSKEYS" POC

6.1 Software architecture

The core of our prototype is based on SoftFIDO [13], an open source FIDO2 software authenticator, which we forked and modified to delegate private key operations to an external back-end [28]. The backend consists of a modified version of SoftHSMv2 [31], an open source PKCS#11 compatible software HSM, extended in our fork [29] to integrate with the Fireblocks mpc-lib cryptographic engine ([16]). The overall architecture is depicted in Figure 1.

6.1.1 Registration. In the PoC, when a credential is created through the WebAuthn registration ceremony, the softfido component triggers the `C_GenerateKeyPair` call through PKCS#11. This request is redirected to our patched SoftHSMv2 instance, which in turn invokes mpc-lib to generate a secure ECDSA keypair via MPC protocols.

6.1.2 Authentication. The WebAuthn challenge is signed by invoking `C_Sign`, again routed through the same MPC backend. The mpc-lib performs the signing operation collaboratively using MPC nodes, and the resulting ECDSA signature is returned to the authenticator. For the purpose of this PoC, all MPC participants are instantiated within the same local process.

6.2 Integration and Build Process

To ensure modularity and reproducibility, the prototype integrates three independently built components: a modified software authenticator [28], a patched PKCS#11 backend [29], and the `mpc-lib` [16] from Fireblocks. The PoC has been successfully built and run on an Ubuntu 24.04 machine.

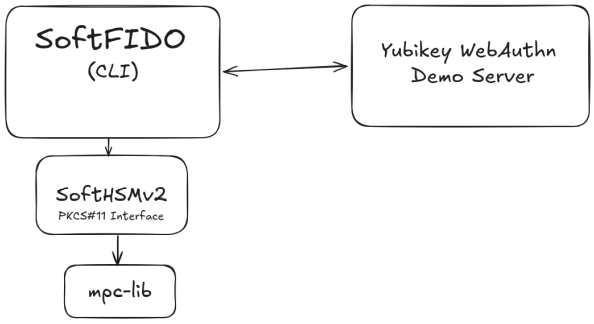

Figure 1: Software Architecture

(1) The `mpc-lib` must first be cloned and built by following the instructions provided in its public repository. Once built, two environment variables must be defined: `MPC_LIB_BUILD_LIB` (pointing to the directory containing the compiled `libcosigner.so`, typically `mpc-lib/build/src/common`) and `MPC_LIB_INCLUDE` (pointing to the `include/` directory of `mpc-lib`).
(2) The patched version of `SoftHSMv2` [29] integrates the MPC backend via dynamic linking, enabling PKCS#11-compatible applications to offload key generation and signing operations to the MPC engine. This was compiled as a shared library. We also needed to define new PKCS#11 constants to define both MPC key generation and MPC signature algorithms as follows[4]:

```
#define CKM_MPCECDSA_KEY_PAIR_GEN
(0x103FUL)
#define CKM_MPCECDSA
(0x104FUL)
```

(3) The authenticator, `SoftFIDO` [28], is then built and configured to interface with the patched PKCS#11 module. The build system includes scripts to fetch, patch, and configure all submodules, ensuring a coherent and reproducible build environment. WebAuthn registration and authentication can then be performed against public test endpoints such as the Yubico WebAuthn Demo Server [1], verifying the correct end-to-end operation of MPC-secured credentials.

6.3 Running the Demo

6.3.1 Registering on the Yubico Demo Server. The following are screenshots of an execution of the PoC. Figure 2 (page 19) shows the home page of the Yubico's Demo Server. Figure 3 (page 19) shows the console logs of the FIDO's authenticator when MPC key generation is triggered. Different colors were used in the console logs to emphasize the computations performed by each of the cosigners. Eventually, Figure 4 (page 20) shows the success of the registration process.

[4]Note that we chose unused values to define our constants, this might break in future releases of PKCS#11

6.3.2 Authenticating on the Yubico Demo Server. The following figures show the execution of a successful authentication on the Yubico Demo Server. Figure 5 (page 20) shows the console logs of the challenge signature during the authentication process. Figure 6 (page 20) shows the displayed page in case of successful authentication.

ACKNOWLEDGMENTS

The authors used generative AI-based tools to revise the text, improve flow and correct typos and grammatical errors.

REFERENCES

[1] [n. d.]. Yubico WebAuthn Demo Server. https://demo.yubico.com/webauthn-technical/registration. Accessed: 2025-04-14.

[2] Michael Adjedj, Constantin Blokh, Geoffroy Couteau, Arik Galansky, Antoine Joux, and Nikolaos Makriyannis. 2024. Two-Round 2PC ECDSA at the Cost of 1 OLE. Cryptology ePrint Archive, Report 2024/1950. https://eprint.iacr.org/2024/1950

[3] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. 2001. Resettably-Sound Zero-Knowledge and its Applications. In *42nd FOCS*. IEEE Computer Society Press, 116–125. https://doi.org/10.1109/SFCS.2001.959886

[4] Mihir Bellare and Gregory Neven. 2006. Multi-signatures in the plain public-Key model and a general forking lemma. In *ACM CCS 2006*, Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati (Eds.). ACM Press, 390–399. https://doi.org/10.1145/1180405.1180453

[5] Constantin Blokh, Nikolaos Makriyannis, and Udi Peled. 2022. Efficient Asymmetric Threshold ECDSA for MPC-based Cold Storage. Cryptology ePrint Archive, Report 2022/1296. https://eprint.iacr.org/2022/1296

[6] Dan Boneh, Iftach Haitner, and Yehuda Lindell. 2024. Exponent-VRFs and Their Applications. Cryptology ePrint Archive, Paper 2024/397. https://eprint.iacr.org/2024/397

[7] Dan Boneh, Iftach Haitner, Yehuda Lindell, and Gil Segev. 2025. Exponent-VRFs and Their Applications *(LNCS)*. Springer, Cham, 195–224. https://doi.org/10.1007/978-3-031-91098-2_8

[8] Dan Boneh, Iftach Haitner, Yehuda Lindell, and Gil Segev. 2025. Exponent-VRFs and Their Applications. private communication.

[9] Thanh Bui, Siddharth Prakash Rao, Markku Antikainen, Viswanathan Manihatty Bojan, and Tuomas Aura. 2018. Man-in-the-Machine: Exploiting Ill-Secured Communication Inside the Computer. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 1511–1525. https://www.usenix.org/conference/usenixsecurity18/presentation/bui

[10] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. 2020. UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 1769–1787. https://doi.org/10.1145/3372297.3423367

[11] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. 2021. UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts. Cryptology ePrint Archive, Report 2021/060. https://eprint.iacr.org/2021/060

[12] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. 2024. Threshold ECDSA in Three Rounds. In *2024 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 3053–3071. https://doi.org/10.1109/SP54263.2024.00178

[13] Ellerh. 2024. softfido. https://github.com/ellerh/softfido.

[14] FIDO Alliance. 2025. FIDO Alliance: Changing the Nature of Authentication. https://fidoalliance.org/. The FIDO Alliance is an open industry association working to reduce the world's reliance on passwords by developing and promoting open standards for authentication. The website features sections such as Passkey Central, Authenticate Conference, Alliance Membership, Specifications, Certification, Device Onboarding, and more. Accessed: April 10, 2025..

[15] FIDO Alliance. 2025. Passkey Security – Introduction to Passkeys. https://www.passkeycentral.org/introduction-to-passkeys/passkey-security. This web page from Passkey Central details how passkeys improve security by offering phishing resistance, eliminating server-side secrets, and enabling multi-factor authentication using biometrics and device-bound security keys. Accessed: April 10, 2025..

[16] Fireblocks. 2025. mpc-lib. https://github.com/fireblocks/mpc-lib.

[17] François Garillot, Yashvanth Kondi, Payman Mohassel, and Valeria Nikolaenko. 2021. Threshold Schnorr with Stateless Deterministic Signing from Standard Assumptions. In *CRYPTO 2021, Part I (LNCS, Vol. 12825)*, Tal Malkin and Chris Peikert (Eds.). Springer, Cham, Virtual Event, 127–156. https://doi.org/10.1007/978-3-030-84242-0_6

[18] Mohammed Jubur, Prakash Shrestha, Nitesh Saxena, and Jay Prakash. 2021. Bypassing Push-based Second Factor and Passwordless Authentication with Human-Indistinguishable Notifications. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security* (Virtual Event, Hong Kong) *(ASIA CCS '21)*. Association for Computing Machinery, New York, NY, USA, 447–461. https://doi.org/10.1145/3433210.3453084

[19] Chelsea Komlo and Ian Goldberg. 2020. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. In *SAC 2020 (LNCS, Vol. 12804)*, Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn (Eds.). Springer, Cham, 34–65. https://doi.org/10.1007/978-3-030-81652-0_2

[20] Yashvanth Kondi, Claudio Orlandi, and Lawrence Roy. 2023. Two-Round Stateless Deterministic Two-Party Schnorr Signatures from Pseudorandom Correlation Functions. In *CRYPTO 2023, Part I (LNCS, Vol. 14081)*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer, Cham, 646–677. https://doi.org/10.1007/978-3-031-38557-5_21

[21] Yashvanth Kondi, Claudio Orlandi, and Lawrence Roy. 2023. Two-Round Stateless Deterministic Two-Party Schnorr Signatures from Pseudorandom Correlation Functions. In *Advances in Cryptology – CRYPTO 2023*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer Nature Switzerland, Cham, 646–677.

[22] Dhruv Kuchhal, Muhammad Saad, Adam Oest, and Frank Li. 2023. Evaluating the Security Posture of Real-World FIDO2 Deployments. In *ACM CCS 2023*, Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda (Eds.). ACM Press, 2381–2395. https://doi.org/10.1145/3576915.3623063

[23] Dhruv Kuchhal, Muhammad Saad, Adam Oest, and Frank Li. 2023. Evaluating the Security Posture of Real-World FIDO2 Deployments. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 2381–2395. https://doi.org/10.1145/3576915.3623063

[24] Yehuda Lindell. 2017. Fast Secure Two-Party ECDSA Signing. In *CRYPTO 2017, Part II (LNCS, Vol. 10402)*, Jonathan Katz and Hovav Shacham (Eds.). Springer, Cham, 613–644. https://doi.org/10.1007/978-3-319-63715-0_21

[25] Ahmed Tanvir Mahdad, Mohammed Jubur, and Nitesh Saxena. 2024. Breaching Security Keys without Root: FIDO2 Deception Attacks via Overlays exploiting Limited Display Authenticators. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 1686–1700. https://doi.org/10.1145/3658644.3690286

[26] Nikolaos Makriyannis. 2022. On the Classic Protocol for MPC Schnorr Signatures. Cryptology ePrint Archive, Report 2022/1332. https://eprint.iacr.org/2022/1332

[27] Megaloblastt. [n. d.]. Proof-of-Concept Passkeys MPC. https://github.com/Megaloblastt/passkeys_mpc.

[28] Megaloblastt. 2025. Modified softfido for MPC integration. https://github.com/Megaloblastt/softfido.

[29] Megaloblastt. 2025. Modified SoftHSMv2 with MPC backend. https://github.com/Megaloblastt/SoftHSMv2.

[30] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. 2020. MuSig-DN: Schnorr Multi-Signatures with Verifiably Deterministic Nonces. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 1717–1731. https://doi.org/10.1145/3372297.3417236

[31] OpenDNSSEC Project. 2024. SoftHSMv2. https://github.com/softhsm/SoftHSMv2.

[32] Mario Rivas, Ruben Santos, and Jorge Sanz. 2025. In-Depth Technical Analysis of the Bybit Hack. https://www.nccgroup.com/uk/research-blog/in-depth-technical-analysis-of-the-bybit-hack/. Blog post published by NCC Group.

[33] Peng Xu, Ruijie Sun, Wei Wang, Tianyang Chen, Yubo Zheng, and Hai Jin. 2021. SDD: A trusted display of FIDO2 transaction confirmation without trusted execution environment. *Future Gener. Comput. Syst.* 125, C (Dec. 2021), 32–40. https://doi.org/10.1016/j.future.2021.06.034

A OPTIMIZATION FOR AVAILABILITY

We revisit the problem from Section 1.1.4. We note that it suffices to minimize the value of t and k for some target probability p under the constraint

$$S_{\varepsilon,\alpha}(t,k) = \varepsilon^t + \sum_{i=1}^{t} \varepsilon^{t-i}\alpha^{ik} < p$$

To see why, it is easy to see that $\Pr[E] \leq S_{\varepsilon,\alpha}(t,k)$ where E denotes the event that availability fails for a given client. The claim below allows us to find the optimal t and k.

Claim A.1. *Fix $t > \log(p)/\log(\varepsilon)$. The equation $f(r) = \mu$ for $f(r) = (r^{t+1}-1)/(r-1)$ and $\mu = \varepsilon^{-t}p$ admits a unique solution $r_0 \in \mathbb{R}$. Furthermore, $S_{\varepsilon,\alpha}(t,k) \leq p$ is equivalent to $k \geq \log(\varepsilon \cdot r_0)/\log(\alpha)$.*

Table 3: Possible values for (t,k) for fixed ε, α and p

(α, ε, p)	Possible values for (t,k)
$(0.5, 10^{-2}, 10^{-5})$	$(t,k) \in \{(3,6),(4,5),(5,4),(6,3)\}$
$(0.1, 10^{-2}, 10^{-5})$	$(t,k) \in \{(3,2),(4,2),(5,2),(6,1)\}$
$(0.5, 10^{-3}, 10^{-5})$	$(t,k) \in \{(2,9),(3,6),(4,5),(5,4)\}$
$(0.1, 10^{-3}, 10^{-5})$	$(t,k) \in \{(2,3),(3,2),(4,2),(5,2)\}$
$(0.5, 10^{-2}, 10^{-6})$	$(t,k) \in \{(4,6),(5,5),(6,4),(7,3)\}$
$(0.1, 10^{-2}, 10^{-6})$	$(t,k) \in \{(4,2),(5,2),(6,2),(7,1)\}$
$(0.5, 10^{-3}, 10^{-6})$	$(t,k) \in \{(2,64),(3,7),(4,5),(5,4)\}$
$(0.1, 10^{-3}, 10^{-6})$	$(t,k) \in \{(2,20),(3,3),(4,2),(5,2)\}$
$(0.5, 10^{-2}, 10^{-9})$	$(t,k) \in \{(5,7),(6,6),(7,5),(8,4)\}$
$(0.1, 10^{-2}, 10^{-9})$	$(t,k) \in \{(5,2),(6,2),(7,2),(8,2)\}$
$(0.5, 10^{-3}, 10^{-9})$	$(t,k) \in \{(3,64),(4,8),(5,6),(6,5)\}$
$(0.1, 10^{-3}, 10^{-9})$	$(t,k) \in \{(3,20),(4,3),(5,2),(6,2)\}$

Proof. First, we note that f is a non-decreasing function of r. Indeed, $f(r) = 1 + r + \ldots + r^t$ is clearly non-decreasing for $r \geq 0$. As such, for any $\mu > 1$, it admits a unique solution to the equation $f(r) = \mu$. Finally,

$$\begin{aligned} k \geq \log(\varepsilon \cdot \mu)/\log(\alpha) \quad &\Leftrightarrow \\ \alpha^k/\varepsilon \leq \mu \quad &\Leftrightarrow \\ f(\alpha^k/\varepsilon) \leq f(r_0) \quad &\Leftrightarrow \\ S_{\varepsilon,\alpha}(t,k)/\varepsilon^t \leq \mu \quad &\Leftrightarrow \\ S_{\varepsilon,\alpha}(t,k)/\varepsilon^t \leq p/\varepsilon^t \quad &\Leftrightarrow \\ S_{\varepsilon,\alpha}(t,k) \leq p \end{aligned}$$

□

We conclude by providing concrete numerical values in Table 3.

B UC POLICY-COMPLIANT SIGNATURES

B.1 UC framework reminder

We use the simplified variant of the UC framework. In this section we provide a quick reminder of the framework.

B.1.1 The model for two-party protocols Π. For the purpose of modeling the protocols in this work, we consider a system that consists of the following 4 *machines,* where each machine is a computing element (say, an interactive Turing machine) with a specified program and and identity. First, we have two machines with program Π and identities $\mathsf{P}_1, \mathsf{P}_2$. Next, we have a machine $\mathcal{A}$ representing the adversary and a machine $\mathcal{Z}$ representing the environment. All machines are initialized on a security parameter κ and are polynomial in κ. The environment $\mathcal{Z}$ is activated first, with an external input z. $\mathcal{Z}$ activates the parties, chooses their input and reads their output. $\mathcal{A}$ can corrupt parties and instruct them to leak information to $\mathcal{A}$ and to perform arbitrary instructions. $\mathcal{Z}$ and $\mathcal{A}$ communicate freely throughout the computation. The real process terminates when the environment terminates. Let $\mathrm{EXEC}^{\mathcal{Z}}_{\Pi,\mathcal{A}}(1^\kappa, z)$ denote the environment's output in the above process.

B.1.2 Ideal Process. the ideal process is identical to the real process, with the exception that now the machines $\mathsf{P}_1, \mathsf{P}_2$ do not run Π, Instead, they all forward all their inputs to a subroutine machine, called the *ideal functionality* F. Functionality F then processes all the inputs locally and returns outputs to $\mathsf{P}_1, \mathsf{P}_2$. Let $\mathrm{EXEC}^{\mathcal{Z}}_{\mathsf{F},\mathsf{S}}(1^\kappa, z)$ denote the environment's output in the above process.

Definition B.1. We say that Π UC-realizes F if for every adversary $\mathcal{A}$ there exists a simulator S such that for every environment $\mathcal{Z}$ it holds that

$$\{\mathrm{EXEC}^{\mathcal{Z}}_{\Pi,\mathcal{A}}(1^\kappa, z)\}_{z\in\{0,1\}^*} \equiv \{\mathrm{EXEC}^{\mathcal{Z}}_{\mathsf{F},\mathsf{S}}(1^\kappa, z)\}_{z\in\{0,1\}^*}$$

B.1.3 The Adversarial Model. The adversary can corrupt parties statically throughout the computation and the corruption status of a given party does not change thereafter. Once corrupted, the party reports all its internal state to the adversary, and from now on follows the instructions of the adversary. We also allow the adversary to *reset* honest parties to a previous round during an execution of protocol Π. In our model, parties are assumed to have a dedicated input tape for reset attacks. By writing on this special tape at designated stages of the protocol, the adversary can reset the state of an honest party to that of an earlier round, as if later messages had not been exchanged. After the reset, the honest party continues execution with the same inputs and random tape.

B.2 Ideal Functionality

Functionality $\mathsf{F}_{\mathrm{PolicySig}}$

Initialization:

At first activation by party U, interpret the given session ID as a tuple $sid = (\ldots, \mathsf{P}, \mathsf{W})$, where P is the *policy* and W is the *service provider.* Record U and W as `non-corrupted`, and initialize a state variable to `uncompromised`.

Key-generation:

(1) Upon receiving (init) from $\mathsf{X} \in \{\mathsf{U}, \mathsf{W}\}$, send (init, X) to S.
(2) Upon receiving (pubkey, V) from S, if all U and W have already input init, and no tuple (pubkey, V′) is recorded, then record (pubkey, V) and output (pubkey, V) to U.

Signing:

(1) Upon receiving (sign, m) from $\mathsf{X} \in \{\mathsf{U}, \mathsf{W}\}$:
  - If both U and W have provided a (sign, m) input and $\mathsf{P}(m) = 1$ then record (m). '*Policy Compliance*'
  - Send (sign, m, X) to S.
(2) Upon receiving (sig-string, m, σ) from S:
  - If (m) is not recorded or (m, σ) is already recorded, then end the activation.
  - Else, set $\beta = \mathsf{V}(m, \sigma)$ and record (m, σ).

  Output (sig-string, m, σ) to U.

Verification:

Upon receiving $(\text{sig-vrfy}, m, \sigma, \mathsf{V}')$ from a party Q, do:
- If $\mathsf{V}' = \mathsf{V}$, no record of (m) among previously recorded messages, the state is `uncompromised`, then output $(\text{istrue}, m, \sigma, 0)$ to Q. '*Unforgeability*'
- Else output $(\text{istrue}, m, \sigma, \beta)$ to Q, where $\beta = \mathsf{V}'(m, \sigma)$. '*Consistency*'

Corruption:

(1) Upon receiving input $(\text{corrupt}, \mathsf{X})$ from S, record $\mathsf{X} \in \{\mathsf{U}, \mathsf{W}\}$ as (`corrupted`). If both U and W are recorded as corrupted then change state to `compromised`.
(2) Upon receiving input (status) from $\mathsf{X} \in \{\mathsf{U}, \mathsf{W}\}$, return to X its corruption status.

Functionality 1: Ideal Policy-Compliant Signing

B.2.1 The session identifier mechanism. A key design decision in our ideal functionality is how sessions are identified. The ideal functionality uses a single, unique session identifier *sid* that captures the participating parties and other long-term contextual data, and is essentially tied to the public key. Notably, there are no session- or message-specific identifiers during signing. This is intentional: introducing such identifiers would break the statelessness property we aim to achieve. Designing protocols that remain composable without relying on additional identifiers is non-trivial. It is easy to devise protocols that are secure in a setting where all subprotocols have session identifiers, yet completely insecure under reset attacks. The functionality provides strong unforgeability guarantees—it only accepts signatures if the messages have been submitted by both parties and comply with the agreed-upon security policy. The challenge lies in proving that our protocols realize this functionality.

B.2.2 Strong unforgeability. Unavoidably, the functionality does not support *strong unforgeability*, meaning that an attacker can produce a new valid signature for a message that has already been signed. Since the honest party does not remember whether it has signed a given message, an attacker can inject fresh randomness into the signing process, resulting in a different signature for the same message—thus violating strong unforgeability. If this property is important, it can be obtained by enforcing determinism on both parties and making the corresponding adjustments to the ideal functionality. We do not enforce determinism by default, as our primary goal is to preserve statelessness and maximize scalability, and full determinism is unnecessary for our targeted security guarantees.

B.3 Resettable Security to UC reduction

In this section, we show that signing protocols Σ that are secure against reset attacks in the sense of Definition 3.2 give rise to protocols that UC-realize $\mathsf{F}_{\text{PolicySig}}$.

Theorem B.2. *Let Σ denote a two-party signing protocol that is secure against reset attacks in the sense of Definition 3.2. Let Σ' denote the variant of Σ such that:*

(1) U and W take the roles of P_1 and P_2 in Σ. U has an additional input P, representing the policy *which it sends to W during key-generation.*
(2) Before generating any signature on message msg*, the parties first check that $\mathsf{P}(\text{msg}) = 1$. If not, the parties abort and do not proceed with the signature generation.*
(3) The protocol is augmented with suitable identifiers so that the input-output behavior of the protocol conforms with the expected input-output behavior of $\mathsf{F}_{\text{PolicySig}}$.

Then, it holds that Σ' UC-realizes $\mathsf{F}_{\text{PolicySig}}$.

Proof. We provide a sketch of the proof, as the proof is very similar to the proofs in [5, 10, 11, 26] showing that the signing protocols UC-realize an the ideal threshold signature functionality. We define the following ideal-world adversary $\mathcal{S}$ that proceeds as follows:

(1) To simulate the honest party, $\mathcal{S}$ simply runs the code of the the honest party according to Σ (without loss of generality, we assume that $\mathcal{Z}$ sends messages on behalf of the corrupted party). When $\mathcal{Z}$ decides to reset the honest party, $\mathcal{S}$ resets its simulation of that party to the specified point and then proceeds according to the protocol.
(2) When interacting with the ideal functionality, $\mathcal{S}$ forwards the verification key V (obtained at the end of the key-generation phase) and the signatures σ (obtained at the end of each signing instance) as provided from the interaction in Item 1.

It remains to argue that $\mathcal{Z}$ cannot distinguish the above simulation strategy from an execution of the protocol in the real world. First, note that $\mathcal{Z}$'s view of the protocol in the real and ideal worlds is identically distributed. This is because $\mathcal{S}$ simply runs the code of the honest party when interacting with $\mathcal{Z}$, which by definition is indistinguishable from a real-world execution. So, the only way for the environment to distinguish between the real and ideal executions is if a forgery is produced—in the real world, forgeries would be accepted as valid signatures, whereas in the ideal world, they are rejected by the functionality. However, this contradicts the presumed security of Σ (which implies the security of the variant Σ'). This concludes the proof. □

C PRELIMINARIES FOR THE WEAKLY-VERIFIABLE EVRF

For $a, b \in \mathbb{N}$, we write $a \mid b$ for 'a divides b' and $a \nmid b$ for the negation. We write $\gcd : \mathbb{N}^2 \to \mathbb{N}$ for the greatest common divisor operation, $[a]_q$ denotes the modular reduction operation $a \bmod q$, and $\varphi(\cdot)$ denotes Euler's totient function.

C.1 Security Assumptions

In the following, we denote by $\text{SampleRSA}(1^\ell)$ an algorithm that returns an RSA modulus $N = p \cdot q$ where $p = q = 3 \bmod 4$, $\gcd(p-1, q-1) = 2$ and $|p| = |q| \geq n_{\text{rsa}}/2 - 1$. In our concrete runtime estimates, we use the optimized RSA sampling algorithm introduced in [2]. This algorithm is guaranteed to produce a 2^ℓ-*tough RSA modulus*, as defined below:

Definition C.1 (Tough RSA moduli)**.** Let $N = pq$ denote an RSA modulus. For $\ell \in \mathbb{N}$, we say that N is 2^ℓ-tough if $\gcd(p-1, q-1) = 2$ and $u \mid (p-1)/2$ (resp. $u \mid (q-1)/2$) implies $u > 2^\ell$.

We recall the strong-RSA assumption below.

Definition C.2 (SRSA)**.** Define distribution ensemble SRSA such that $(N, c) \xleftarrow{\$} \mathsf{SRSA}(1^\ell)$ for $N \xleftarrow{\$} \mathsf{SampleRSA}(1^\ell)$ and $c \xleftarrow{\$} \mathbb{Z}_N^*$. We say that *strong-RSA holds true* if for every PPTM A, there exists a negligible function $\mu : \mathbb{N} \to \mathrm{R}$ such that

$$\Pr\left[(e, x) \xleftarrow{\$} \mathsf{A}(1^\ell, N, c) \;\text{ s.t. }\; c = x^e \bmod N \wedge e \notin \{-1, 1\}\right] \leq \mu(\ell),$$

where the probability is over the choice of $(N, c) \xleftarrow{\$} \mathsf{SRSA}(1^\ell)$.

We will rely on the following lemma (Lemma 3.13 from [2]), slightly simplified to hold w.r.t. a single basis instead of two:

Lemma C.3. *Let N denote a 2^ℓ-tough RSA modulus and let $g \in \mathrm{QR}_N$ such that $\langle g \rangle = \mathrm{QR}_N$. Suppose there exists PPTM* A *such that for some $\varepsilon \geq 0$:*

$$\Pr\left[(x, y, e, C) \xleftarrow{\$} \mathsf{A}(N, g, h) \;\; s.t. \;\; g^x h^y = C^e \wedge |e| < 2^\ell \wedge e \not| \, y\right] \geq \varepsilon$$

where the probability is over the choice of $h \xleftarrow{\$} \mathrm{QR}_N$. Then, there exists PPTM B *such that*

$$\Pr\left[(c, R) \xleftarrow{\$} \mathsf{B}(N, g) \;\; s.t. \;\; g = R^c \wedge c \notin \{-1, 1\}\right] \geq \varepsilon \cdot \left(2 \cdot \frac{\varphi(N)}{N-1} - \frac{3}{2}\right)$$

We also rely on the short-exponent indistinguishability assumption:

Definition C.4 (SEI over $\mathbb{Z}_N^*$)**.** Define distribution ensemble SEI such that $(N, t, [t^{\alpha x + (1-\alpha) y}]_N, \alpha) \xleftarrow{\$} \mathsf{SEDL}(1^\ell)$ for $N \xleftarrow{\$} \mathsf{SampleRSA}(1^\ell)$, $t \xleftarrow{\$} \mathrm{QR}_N$, $x \xleftarrow{\$} \pm 2^{2\ell}$, $y \xleftarrow{\$} \pm N$ and $\alpha \xleftarrow{\$} \{0, 1\}$. We say that *small-exponent indistinguishability (SEI) holds true* if for every PPTM A, there exists a negligible function $\mu : \mathbb{N} \to \mathrm{R}$ such that

$$\Pr_{(N,t,s,\alpha) \xleftarrow{\$} \mathsf{SEI}(1^\ell)}\left[\alpha_0 \xleftarrow{\$} \mathsf{A}(1^\ell, N, t, s) \;\text{ s.t. }\; \alpha_0 = \alpha\right] \leq \frac{1}{2} + \mu(\ell).$$

In [2], it was shown that SEI over $\mathbb{Z}_N^*$ reduces to the short-exponent discrete logarithm assumption, *i.e.*, the hardness of solving discrete logs over $\mathbb{Z}_N^*$ for random 2ℓ-bit-long exponents.

Definition C.5 (DCR)**.** Define DCR such that $(N, [\rho^{\alpha N} \cdot c^{1-\alpha}]_{N^2}, \alpha) \xleftarrow{\$} \mathsf{DCR}(1^\ell)$ for $N \xleftarrow{\$} \mathsf{SampleRSA}(1^\ell)$, $\rho \xleftarrow{\$} \mathbb{Z}_N^*$, $c \xleftarrow{\$} \mathbb{Z}_{N^2}^*$ and $\alpha \xleftarrow{\$} \{0, 1\}$. We say that *decisional composite residuosity (DCR) holds true* if for every PPTM A, there exists a negligible function $\mu : \mathbb{N} \to \mathrm{R}$ such that

$$\Pr_{(N,x,\alpha) \xleftarrow{\$} \mathsf{DCR}(1^\ell)}\left[\alpha_0 \xleftarrow{\$} \mathsf{A}(1^\ell, N, x) \;\text{ s.t. }\; \alpha_0 = \alpha\right] \leq \frac{1}{2} + \mu(\ell).$$

C.2 Paillier Encryption & Damgård-Fujisaki Commitments

Definition C.6. Paillier Encryption consists of the following algorithms.

(1) *KeyGen.* On input $\ell \in \mathbb{N}$, sample $(N, n_{\mathrm{rsa}}; p, q) \xleftarrow{\$} \mathsf{SampleRSA}(1^\ell)$.
Return $(N; \varphi(N))$. (Optionally, output $\rho_0 \xleftarrow{\$} \mathbb{Z}_N^*$.)

(2) *Encrypt.* On input $m \in \mathbb{Z}_N$ and N, sample $\rho \xleftarrow{\$} \mathbb{Z}_N^*$ and output
$$C = \mathsf{enc}_N(m; \rho) = (1 + mN) \cdot \rho^N \bmod N^2.$$

(3) *Decrypt.* On input $C \in \mathbb{Z}_{N^2}^*$ and $\varphi(N)$, output $m = \left(\frac{[C^{\varphi(N)}]_{N^2} - 1}{N}\right) \cdot \varphi(N)^{-1} \bmod N$

In this work, we rely on the well-known fact that the IND-CPA security of Paillier reduces to the DCR assumptions.

Definition C.7. Damgård-Fujisaki commitments in RSA groups consist of the following algorithms.

(1) *KeyGen.* On input $\ell \in \mathbb{N}$, sample $(\hat{N}, n_{\mathrm{rsa}}, v, u) \xleftarrow{\$} \mathsf{SampleDF}(1^\ell)$. Optionally, output λ such that $v^\lambda = u \bmod \hat{B}$.

(2) *Commit.* For $m \in \mathbb{Z}$ and $(\hat{M}, v, u)$, sample $\rho \xleftarrow{\$} [\hat{M} \cdot 2^\ell]$ and output
$$\widetilde{C} = u^m \cdot v^\rho \bmod \hat{M}.$$

C.3 Zero-knowledge proofs

Our protocol makes uses NIZK arising from Fiat-Shamir-compiled zero knowledge protocols for which we give a refresher. Let $\boldsymbol{R} = \{(X; w)\}$ denote an NP-relation. We define Σ-protocol Π a four-tuple of algorithms $(\mathsf{S}, \mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ parametrized by $\mathbf{E}$ and $\boldsymbol{T}$ such that: (i) $\sigma \xleftarrow{\$} \mathsf{S}(1^\ell)$ denotes the setup parameter. (ii) $A \leftarrow \mathsf{P}_1(\sigma, X; w, \tau)$ denotes the provers' first message where $\tau \in \boldsymbol{T}$. (iii) $z \xleftarrow{\$} \mathsf{P}_2(\sigma, X, e; w, \tau)$ denotes the provers' second message where $e \in \mathbf{E}$. (iv) $\beta \leftarrow \mathsf{V}(\sigma, X, A, e, z)$ denotes the verifiers' acceptance/rejection. The tuple (σ, X, A, e, z) is called the transcript of the protocol. Additionally Π is secure if it satisfies the soundness and zero-knowledge properties with respect to $\boldsymbol{R}$, outlined below.

C.3.1 Σ-protocols. Σ-protocols are 3-move public-coin honest-verifier zero-knowledge proof systems. Formally, a Σ-protocol is a four-tuple $(\mathsf{S}, \mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ with challenge space $\mathbf{E}$ and commitment space $\boldsymbol{T}$ where $\sigma \xleftarrow{\$} \mathsf{S}(1^\ell)$ denotes the setup parameter, $A \leftarrow \mathsf{P}_1(\sigma, X; w, \tau)$ denotes the provers' first message (where $\tau \in \boldsymbol{T}$), $z \leftarrow \mathsf{P}_2(\sigma, X, e; w, \tau)$ denotes the provers' second message (where $e \in \mathbf{E}$), and $\beta \leftarrow \mathsf{V}(\sigma, X, A, e, z)$ denotes the verifiers' acceptance/rejection. The tuple (σ, X, A, e, z) is called the transcript of the protocol Π for an NP-relation $\boldsymbol{R}$ with common input $X \in \boldsymbol{R}$ and secret input x with auxiliary values aux. A Σ-protocol is

- μ-**HVZK** if For any $\sigma \xleftarrow{\$} \mathsf{S}$, there exists H such that the following are μ-statistically close:
- (σ, X, A, e, z) for $\tau \xleftarrow{\$} \boldsymbol{T}$, $e \xleftarrow{\$} \mathbf{E}$, $A \leftarrow \mathsf{P}_1(\sigma, X; w, \tau)$, $z \leftarrow \mathsf{P}_2(\sigma, X, e; w, \tau)$
- (σ, X, A', e', z') for $e' \xleftarrow{\$} \mathbf{E}$, $(A', z') \xleftarrow{\$} \mathsf{H}(\sigma, X, e')$ such that $\mathsf{V}(\sigma, X, A', e', z') = 1$.
- **Sound** if for $(X; w) \notin \boldsymbol{R}$, then the following holds for every $(\mathsf{P}_1^*, \mathsf{P}_2^*)$:

$$\Pr_{\substack{e \xleftarrow{\$} \mathbf{E}, \sigma \xleftarrow{\$} \mathsf{S}(1^\ell) \\ A \xleftarrow{\$} \mathsf{P}_1^*(\sigma, X)}}\left[z \xleftarrow{\$} \mathsf{P}_2^*(\sigma, X, e) \;\text{ s.t. }\; \mathsf{V}(\sigma, X, A, e, z) = 1\right] \leq \lambda$$

C.3.2 Fiat-shamir. We use the Fiat-Shamir transform for compiling Σ-protocols into NIZKs in the random oracle model. In particular, for a ZK protocol Π with common input X and secret input x, we write $\psi \xleftarrow{\$} \Pi^{\mathrm{FS}}(\mathrm{aux}, X; x)$ for the transcript of the protocol resulting from the Fiat-Shamir transform. Namely, the verifier's second round

message is calculated as $e = \mathsf{H}(\text{aux}, X, A)$, where A denotes the prover's first-round message, aux denotes auxiliary contextual data and HASH denotes the random oracle. We write $\mathsf{vrfy}_{\Pi^{\text{FS}}}(\text{aux}, x, \pi)$ for the output of the verification algorithm associated with Π^{FS} on common input X, proof ψ and auxiliary input aux.
we let $\mathsf{Sim}_{\boldsymbol{R}}$ for $\Pi^{\text{FS}}_{\boldsymbol{R}}$ that is given programming access to the random oracle $\mathsf{H} : \{0,1\}^* \to \mathbf{E}$: on input (σ, X), $\mathsf{Sim}_{\boldsymbol{R}}$ samples $e' \xleftarrow{\$} \mathbf{E}$ and computes (A', z') by running the HVZK simulator on input (σ, X, e'). If (σ, X, A') was never queried before, Sim programs $\text{HASH}(\sigma, X, A') = e'$ and outputs (A', e', z'); otherwise, it outputs $\perp$. In this work, we consider Σ-protocols satisfying a mild additional condition (satisfied by all protocols used in this work): the min-entropy of the distribution $\{A' : (A', e', z') \xleftarrow{\$} \mathsf{Sim}_{\boldsymbol{R}}(\sigma, X)\}$ is at least 2ℓ. The following fact from [2, Remark 5.2] is folklore:

Fact C.8. *Let $\Pi_{\boldsymbol{R}}$ denote a μ-HVZK Σ-protocol for a relation $\boldsymbol{R}$ satisfying the above additional condition. The advantage of any Q-query adversary in distinguishing $\psi \xleftarrow{\$} \Pi^{\text{FS}}(\text{aux}, X; x)$ from $\psi' \xleftarrow{\$} \mathsf{Sim}_{\boldsymbol{R}}(\sigma, X)$ is at most $Q/2^{2\ell} + \mu$.*

C.3.3 Generalized Forking Lemma. To analyze the soundness of zero-knowledge proofs compiled into NIZKs via the Fiat-Shamir transform, we rely on the generalized forking lemma from [4]. The lemma below is reproduced verbatim from [2]:

Lemma C.9 (Generalized Forking Lemma [4]). *Fix a bound $Q \geq 1$ and a set S of size $|S| \geq 2$. Let $\mathcal{A}$ be a randomized algorithm that, on input $(x, h_1, \cdots, h_Q)$ returns a pair, the first element of which is an integer in the range $\{0, \cdots, Q\}$ and the second element of which we refer to as a side output. Let D_{IG} be a randomized algorithm that we call the input generator. We define*

$$\varepsilon := \Pr\Big[i^* \geq 1 \;\Big|\; x \xleftarrow{\$} \mathsf{D}_{\mathsf{IG}}, (h_1, \ldots, h_Q) \xleftarrow{\$} S^Q, \\ (i^*, \text{so}) \xleftarrow{\$} \mathcal{A}(x, h_1, \ldots, h_Q)\Big].$$

and call ε the accepting probability *of $\mathcal{A}$. Then,*

$$\Pr[b = 1 \mid x \xleftarrow{\$} \mathsf{D}_{\mathsf{IG}}, (b, \text{so}, \text{so}') \xleftarrow{\$} \mathsf{F}_{\mathcal{A}}(x)] \geq \frac{\varepsilon^2}{Q} - \frac{\varepsilon}{|S|},$$

where $\mathsf{F}_{\mathcal{A}}$ denotes the forking algorithm *associated to $\mathcal{A}$ and represented on Algorithm 1.*

Algorithm $\mathsf{F}_{\mathcal{A}}(x)$

Parameters. An integer $Q \geq 1$. A set S of size $|S| \geq 2$. A bound $\mathsf{T}_{\mathcal{A}}$ on the length of the random tape used by $\mathcal{A}$.
Operations.
(1) Sample coins $\rho \xleftarrow{\$} \{0,1\}^{\mathsf{T}_{\mathcal{A}}}$ for $\mathcal{A}$.
(2) Sample $(h_1, \cdots, h_Q) \xleftarrow{\$} S^Q$.
(3) $(i, \text{so}) \xleftarrow{\$} \mathcal{A}(x, h_1, \cdots, h_Q; \rho)$
(4) If $i = 0$, return $(0, \perp, \perp)$.
(5) Sample $(h'_i, \cdots, h'_Q) \xleftarrow{\$} S^{Q-i+1}$.
(6) $(i', \text{so}') \xleftarrow{\$} \mathcal{A}(x, h_1, \cdots, h_{i-1}, h'_i, \cdots, h'_Q; \rho)$.
(7) If $i = i'$ and $h_i \neq h'_i$, return $(1, \text{so}, \text{so}')$.
(8) Else, return $(0, \perp, \perp)$.

Algorithm 1: Forking algorithm associated to $\mathcal{A}$.

C.3.4 NP-relations and zero-knowledge proofs. Our protocols rely on various zero-knowledge proofs that we import from [2].

Definition C.10 (Well-Formed Modulus)**.** Define $\boldsymbol{R}_{\text{mod}} \ni (N; p_1, p_2)$ such that $N = p_1 \cdot p_2$ and $p_1, p_2 \in \mathsf{PRIMES}^{>2}$, $N = 1 \bmod 4$ and $\gcd(\varphi(N), N) = 1$.

We use the two-move statistical honest-verifier zero-knowledge proof Π_{mod} for $\boldsymbol{R}_{\text{mod}}$ from [2]. We rely on the following fact about its security (see Proposition 5.7 and Lemma 5.8 in [2]):

Fact C.11. *If $(p_1, p_2) = (3, 7) \bmod 8$, the protocol Π_{mod} is 0-HVZK. Furthermore,*
(1) If $\gcd(N, \varphi(N)) \neq 1$, then the verifier accepts with probability at most $1/\gcd(N, \varphi(N))$.
(2) If N has more than three factors, then the verifier accepts with probability at most $1/4$.

Definition C.12 (Small Factors)**.** Define $\boldsymbol{R}_{\text{fac}} \ni (N, \ell; u, v)$ such that $N = u \cdot v$ and $u, v \in \pm\sqrt{N} \cdot 2^{\ell}$.

We rely on the protocol Π_{fac} from [2]. Π_{fac} is a Σ-protocol that relies on an auxiliary group QR_d where $d > 2^{n_{\text{rsa}}+2\ell+2\nu+3}$ is a strong prime and QR is the set of quadratic residues over $\mathcal{F}_d$. The protocol uses two auxiliary commitments $\widetilde{P} = g_d^{p_1} h_d^{r_1} \bmod d$ and $\widetilde{Q} = g_d^{p_2} h_d^{r_2} \bmod d$ to the factors (p_1, p_2) of N, where g_d, h_d are random generators of QR_d (where the discrete logarithm of h_d in basis g_d is unknown to the prover) and (r_1, r_2) are random coins. The fact below summarizes Proposition 5.5 and Lemma 5.6 from [2]:

Fact C.13. *Π_{fac} is $2^{-\nu+1}$-HVZK. Furthermore, let $\text{trans} = (A, e, z)$ and $\text{trans}' = (A, e', z')$ denote two accepting transcripts of Π_{fac} with $e \neq e'$. Then, there exists an extractor E such that at least one of the following holds true.*
(1) $\alpha \leftarrow \mathsf{E}(\text{trans}, \text{trans}')$ and $g^{\alpha} = h \bmod d$.
(2) $(u, v) \leftarrow \mathsf{E}(\text{trans}, \text{trans}')$ such that $N = uv$ and $|u|, |v| \leq 2^{n/2+\ell+\nu}$

Definition C.14 (Damgård-Fujisaki Params.)**.** Define $\boldsymbol{R}_{\text{df}} \ni (\hat{N}, u, v; \lambda)$ such that

$$v \in \mathbb{Z}^*_{\hat{N}}, \quad u = v^{\lambda} \bmod \hat{N}, \quad \text{and} \quad \lambda \leq 2^{2\ell+\nu}.$$

We use a straightforward variant of Π_{df} from [2] with a single basis u instead of 2. We recall Proposition 5.18 from [2] (which extends immediately to our simpler setting):

Fact C.15. *Π_{df} is $2^{-\nu+1}$-sound and 2^{-1}-HVZK.*

We will rely on ℓ parallel repetitions of Π_{df} to achieve $2^{-\ell}$ zero-knowledge error, and denote $\Pi^{\text{FS}}_{\text{df}}$ the resulting Fiat-Shamir-compiled NIZK.

D SECURITY ANALYSIS OF THE WEAKLY-VERIFIABLE EVRF

D.1 Corrupted prover

We first consider the case of a corrupted prover. We define a simulator Sim that emulates the role of a honest verifier.

Proof. Sim maintains a list L_{H}, initially empty, of pairs (query, answer) to the oracle H. Then,

- Sim runs $\mathsf{vk}_{\mathsf{V}} \leftarrow \mathsf{VerKG}^{\mathsf{H}}(1^\ell)$ honestly and sends it to the prover. Let us write $\mathsf{vk}_{\mathsf{V}} = (\hat{M}, v, u, \pi_{\mathsf{df}})$.
- Upon receiving a (possibly malformed) key $\mathsf{vk}_{\mathsf{P}}^\star$ from the prover, Sim stores $\mathsf{vk}_{\mathsf{P}}^\star = (N, \rho, \rho_0, \pi_{\mathsf{mod}}, \pi_{\mathsf{fac}})$. It runs $\mathsf{vrfy}_{\Pi_{\mathsf{mod}}^{\mathsf{FS}}}(\mathsf{aux}, N, \pi_{\mathsf{mod}})$ and $\mathsf{vrfy}_{\Pi_{\mathsf{fac}}^{\mathsf{FS}}}(\mathsf{aux}, N, \pi_{\mathsf{fac}})$.
- If both checks succeed, Sim sends init to the functionality.
- Upon receiving a query val from the prover to H, if L_{H} contains a pair $(\mathtt{val}, \mathcal{E}')$, Sim returns $\mathcal{E}'$. Otherwise, Sim queries $(\mathsf{Eval}, sid, \mathtt{val})$ to the eVRF functionality and receives x. Sim samples $\sigma \xleftarrow{\$} \pm\lfloor N/q \rfloor$ and $\mathcal{E}' \xleftarrow{\$} \mathsf{enc}_N(x + \sigma q)$, stores $(\mathtt{val}, \mathcal{E}')$ in L_{H}, and returns $\mathcal{E}'$.
- For every session with session id sid and input val, upon receiving (X, π) with $X \in \mathbb{G}$, Sim parses $\pi = (\sigma, \psi_{\mathsf{dlenc}})$ and checks that $\mathsf{vrfy}_{\Pi_{\mathsf{dlenc}}^{\mathsf{FS}}}(, ssid, \mathsf{P}, \mathbb{G}, g, q, X, \mathcal{E}, N, \rho, n_x, n_\lambda, \hat{M}, v, u, \psi_{\mathsf{dlenc}}) = 1$. If the check passes, it sends $(\mathsf{proceed}, sid, \mathtt{val})$ to the eVRF functionality on behalf of the ideal adversary; otherwise, it sends $(\mathsf{abort}, sid, \mathtt{val})$ to the eVRF functionality.

We prove that Sim's behavior is indistinguishable from that of a honest verifier. Sim emulates vk_{V} honestly; furthermore, Sim emulates the answers of H on val by sampling uniformly random encryptions $\mathcal{E}'$ of $x + \sigma q$, where x is the output of the eVRF functionality to the prover query $(\mathsf{Eval}, sid, \mathtt{val})$, and σ is a random element from $\pm\lfloor N/q \rfloor$. By definition, the eVRF functionality samples $x \xleftarrow{\$} \mathbb{Z}_q$, hence $x + \sigma q$ is statistically close to a uniformly random element over $\mathbb{Z}_N$, and therefore $\mathcal{E}'$ is statistically close to a uniformly random element of $\mathbb{Z}_{N^2}^*$ (hence to its distribution in the honest protocol). Therefore, the only possible difference between the real protocol and the simulation lies in the output of the verifier (for each input val):

- In the real protocol, if ψ_{dlenc} verifies, the verifier outputs X; otherwise it outputs $\perp$.
- In the simulation, if ψ_{dlenc} verifies, the ideal verifier outputs g^x, where x is the output of the eVRF functionality to the prover query $(\mathsf{Eval}, sid, \mathtt{val})$. Otherwise, it outputs $\perp$.

It remains to prove that, when ψ_{dlenc} verifies, both outputs agree with overwhelming probability. The analysis uses the following lemma:

Lemma D.1. *Let $\mathcal{E}' \in \mathbb{Z}_{N^2}^*$ and $\sigma \in \pm N - \sqrt{N}$, and let n_x such that $2^{n_x} < \sqrt{N}$. Assume that $\mathsf{dec}_{\varphi(N)}(\mathcal{E}' \cdot (1 - \sigma N)) \in \pm 2^{n_x}$ and $\sigma = 0 \bmod q$. Then, $\mathsf{dec}_{\varphi(N)}(\mathcal{E}' \cdot (1 - \sigma N)) = \mathsf{dec}_{\varphi(N)}(\mathcal{E}') \bmod q$.*

Proof. Let $w = \mathsf{dec}_{\varphi(N)}(\mathcal{E}' \cdot (1 - \sigma N))$. Observe that $\mathsf{dec}_{\varphi(N)}(\mathcal{E}') = [w + \sigma]_N$. If $|\sigma| \le (N - \sqrt{N})/2$ and $|w| < 2^{n_x - 1}$, then $|w + \sigma| \le |w| + |\sigma| \le (N - \sqrt{N})/2 + 2^{n_x - 1} \le N/2$, and thus $[w + \sigma]_N = w + \sigma$. In conclusion,

$$\begin{aligned}\mathsf{dec}_{\varphi(N)}(\mathcal{E}' \cdot (1 - \sigma N)) = w \bmod q &= w + \sigma \bmod q \\ &= [w + \sigma]_N \bmod q = \mathsf{dec}_{\varphi(N)}(\mathcal{E}') \bmod q\end{aligned}$$

□

Now, by construction of Sim, it holds that $x = [\mathsf{dec}_{\phi(N)}(\mathsf{H}(\mathtt{val}))]_q$. Hence, it remains to show that, when ψ_{dlenc} verifies, it holds that $X = g^x$ with $x = [\mathsf{dec}_{\phi(N)}(\mathsf{H}(\mathtt{val}))]_q$. We prove this by showing that if there exists a PPT oracle adversary that manages to produce (X, π) such that ψ_{dlenc} verifies but X does not satisfy this relation (with some non-negligible probability δ), then the discrete logarithm assumption does not hold over the set QR_d of quadratic residues over $\mathcal{F}_d$, where $d > 2^{n_{\mathrm{rsa}} + 2\ell + 2\nu + 3}$ is a large public strong prime.

Let $\mathcal{A}$ denote a PPT adversary making at most Q_{H} queries to H. Let $\mathsf{vk}_{\mathsf{V}} \leftarrow \mathsf{VerKG}^{\mathsf{H}}(1^\ell)$, and $(\mathsf{vk}_{\mathsf{P}}^\star, \mathtt{val}, (X, \pi)) \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{H}'}(\mathsf{vk}_{\mathsf{V}})$. Set $\mathsf{vk}^\star = (\mathsf{vk}_{\mathsf{P}}^\star, \mathsf{vk}_{\mathsf{V}})$. Parse

- $\mathsf{vk}_{\mathsf{V}} = (\hat{M}, v, u, \pi_{\mathsf{df}})$
- $\mathsf{vk}_{\mathsf{P}}^\star = (N, \rho, \rho_0, \pi_{\mathsf{mod}}, \pi_{\mathsf{fac}})$
- $\pi = (\sigma, \psi_{\mathsf{dlenc}})$

Let us denote FailSim the event $(X \neq g^{\mathsf{dec}_{\varphi(N)}(\mathcal{E}') \bmod q}) \wedge (\mathsf{Verify}^{\mathsf{H}}(\mathsf{vk}^\star, \mathtt{val}, X, \pi) = 1)$ and $\delta = \Pr[\mathsf{FailSim}]$. Now, let Q_{kg} and Q_{ver} denote a bound on the number of queries to H made by $\mathsf{VerKG}^{\mathsf{H}}$ and $\mathsf{Verify}^{\mathsf{H}}$ respectively, and consider the following PPT algorithm $\mathcal{B}$: it receives as input 1^ℓ and a Q-tuple of uniformly random elements $(h_1, \cdots, h_Q) \in (\{0,1\}^\ell)^Q$, where $Q = Q_{\mathsf{H}} + Q_{\mathsf{kg}} + Q_{\mathsf{ver}}$. Then, it runs $\mathsf{vk}_{\mathsf{V}} \leftarrow \mathsf{VerKG}^{\mathsf{H}}(1^\ell)$, $(\mathsf{vk}_{\mathsf{P}}^\star, \mathtt{val}, (X, \pi)) \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{H}'}(\mathsf{vk}_{\mathsf{V}})$, and $\mathsf{Verify}^{\mathsf{H}}((\mathsf{vk}_{\mathsf{P}}^\star, \mathsf{vk}_{\mathsf{V}}), \mathtt{val}, X, \pi)$, by emulating the answers to the (at most) Q queries of VerKG, $\mathcal{A}$, and Verify to H using $(h_1, \cdots, h_Q)$ as its random tape. If Verify returns 1, it outputs $(i^*, \pi_{\mathsf{fac}})$, where $\pi_{\mathsf{fac}} = (A, e, z)$ and i^* is the index such that $e = h_{i^*}$. Else, it outputs $(0, \perp)$. Note that by assumption, Verify returns 1 with probability at least δ.

By the Generalized Forking Lemma (Lemma C.9), there exists an algorithm $\mathsf{F}_{\mathcal{B}}$ that, on input 1^ℓ, runs $\mathcal{B}$ internally twice on two inputs $(1^\ell, h_1, \cdots, h_{i^*}, h_Q)$ and $(1^\ell, h_1, \cdots, h'_{i^*}, h'_Q)$ (with the same random tape), and returns a triple $(b', \mathrm{so}, \mathrm{so}')$ with $b' \in \{0,1\}$ such that

$$\mathsf{p}_{\mathsf{fork}} = \Pr[b' = 1 \mid (b', \mathrm{so}_1, \mathrm{so}'_1) \leftarrow \mathsf{F}_{\mathcal{B}}(1^\ell)] \ge \frac{\delta^2}{Q} - \frac{\delta}{2^{2\ell}},$$

and when $b' = 1$, $(\mathrm{so}_1, \mathrm{so}'_1) = (\pi_{\mathsf{fac}}, \pi'_{\mathsf{fac}})$ where $\pi_{\mathsf{fac}} = (A, e, z$ and $\pi'_{\mathsf{fac}} = (A, e', z')$. Then, by Fact C.13, there exists an extractor E such that

(1) $\alpha \leftarrow \mathsf{E}(\mathrm{so}_1, \mathrm{so}'_1)$ and $g^\alpha = h \bmod d$.
(2) $(u, v) \leftarrow \mathsf{E}(\mathrm{so}_1, \mathrm{so}'_1)$ such that $N = uv$ and $|u|, |v| \le 2^{n_{\mathrm{rsa}}/2 + \ell + \nu}$

The first item contradicts the discrete logarithm assumption over QR_d. Therefore, with probability at least $\mathsf{p}_{\mathsf{fork}} - \mathsf{Adv}^{\mathsf{dlog}}_{\mathsf{F}_{\mathcal{B}}}(1^{n_{\mathrm{rsa}} + 2\ell + 2\nu + 3})$, it must hold that $N = uv$ and $|u|, |v| \le 2^{n_{\mathrm{rsa}}/2 + \ell + \nu}$.

Claim D.2. $\Pr[\mathsf{FailSim} \wedge (N = uv) \wedge (|u|, |v| \le 2^{n/2 + \ell + \nu}) \wedge (\exists p_1, p_2 \in \mathsf{PRIMES}_{>2}, N = p_1 \cdot p_2)] \ge \mathsf{p}_{\mathsf{fork}} - \mathsf{Adv}^{\mathsf{dlog}}_{\mathsf{F}_{\mathcal{B}}}(1^{n + 2\ell + 2\nu + 3}) - Q/2^\ell$, *where the probability is taken over the choice of* $\mathsf{H} \leftarrow \mathcal{H}$, $\mathsf{vk}_{\mathsf{V}} \leftarrow \mathsf{VerKG}^{\mathsf{H}}(1^\ell)$, *and* $(\mathsf{vk}_{\mathsf{P}}^\star, \mathtt{val}, (X, \pi)) \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{H}'}(\mathsf{vk}_{\mathsf{V}})$.

This follows directly from Lemma 5.8 in [2], that shows that (a single repetition of) Π_{mod} is 1/4-sound with respect to the NP-relation "N is a product of two primes", and the Fiat-Shamir compiler applied

to $\ell/2$ parallel repetitions of Π_{mod}. Note that Claim D.2 implies in particular that $\{u, v\} = \{p_1, p_2\}$, hence that neither of p_1, p_2 can be smaller than $2^{n/2-\ell-\nu}$.

Now, consider a new PPT algorithm C: it receives as input 1^ℓ and a Q-tuple of uniformly random elements $(h_1, \cdots, h_Q) \in (\{0,1\}^\ell)^Q$. It first runs $\mathcal{B}$ on input $(1^\ell, h_1, \cdots, h_Q)$. If $\mathcal{B}$ returns $(i^*, \pi_{\mathsf{fac}})$, it records the index j^* such that $e = h_{j^*}$ in $\pi^b_{\mathsf{dlenc}} = (A, e, z)$ (note that π_{dlenc} necessarily verifies when $\mathcal{B}$ does not return $(0, \perp)$). Then, it samples $(h'_{i^*}, \cdots, h'_Q)$ and proceeds as $\mathsf{F}_{\mathcal{B}}$ to obtain a triple. this triple is equal to $(1, \pi_{\mathsf{fac}}, \pi'_{\mathsf{fac}})$, it runs $\mathsf{E}(\pi_{\mathsf{fac}}, \pi'_{\mathsf{fac}})$. If E returns a pair (u, v) such that $N = uv$ and $|u|, |v| \leq 2^{n/2+\ell+\nu}$, it outputs $(j^*, \psi_{\mathsf{dlenc}})$. Else, it outputs $(0, \perp)$.

Let us denote $\gamma = \mathsf{p}_{\mathsf{fork}} - \mathsf{Adv}^{\mathsf{dlog}}_{\mathsf{F}_{\mathcal{B}}}(1^{n+2\ell+2\nu+3}) - Q/2^\ell$. By the Generalized Forking Lemma, there exists an algorithm F_C that, on input 1^ℓ, returns a triple $(b'', \mathsf{so}_2, \mathsf{so}'_2)$ with $b \in \{0,1\}$ such that

$$\mathsf{p}'_{\mathsf{fork}} = \Pr[b'' = 1 \mid (b, \mathsf{so}_2, \mathsf{so}'_2) \leftarrow \mathsf{F}_C(1^\ell)] \geq \frac{\gamma^2}{Q} - \frac{\gamma}{2^{2\ell}},$$

and when $b'' = 1$, $(\mathsf{so}_2, \mathsf{so}'_2) = (\psi_{\mathsf{dlenc}}, \psi'_{\mathsf{dlenc}})$ where $\psi_{\mathsf{dlenc}} = (A, e, z)$ and $\psi'_{\mathsf{dlenc}} = (A, e', z')$. Then, by Lemma C.9, when $b'' = 1$, there exists an extractor E' such that one of the following hold:

1. $(\alpha, \beta, \zeta) \leftarrow \mathsf{E}'(\mathsf{so}_2, \mathsf{so}'_2)$ such that $\delta \nmid \alpha$ and $u^\alpha v^\beta = \tilde{P}^\zeta \bmod \hat{N}$.
2. $k \leftarrow \mathsf{E}'(\mathsf{so}_2, \mathsf{so}'_2)$ such that $k < 2^\ell$ and $\gcd(k, N) \neq 1$.
3. $(x', r) \leftarrow \mathsf{E}'(\mathsf{so}_2, \mathsf{so}'_2)$ such that
   (a) $(x', r) \in \pm 2^{n_x+\nu+1} \times \mathbb{Z}_N$,
   (b) $X = g^{x'} \in \mathbb{G}$ and $\mathcal{E} = (1+N)^{x'} r^N \bmod N^2$

Claim D.3. *The probability that Item 1 happens is at most* $\varepsilon = (\mathsf{Adv}^{\mathsf{sei}}_{\mathsf{F}_C}(1^\ell) \cdot (2\phi(N)/(N-1) - 3/2)^{-1} + 2^{-\nu+1} + Q_{\mathsf{H}}^2/2^{2\ell}$.

Proof. Assume that Item 1 happens with probability ε. We modify the game as follows: instead of computing $\mathsf{vk}_{\mathsf{V}} \leftarrow \mathsf{VerKG}^{\mathsf{H}}(1^\ell)$, we pick an RSA modulus $\hat{M}$, sample $u, v \xleftarrow{\$} \mathsf{QR}_N$, and set $\pi_{\mathsf{df}} \xleftarrow{\$} \mathsf{Sim}^{\mathsf{FS}}_{\mathsf{df}}(\mathsf{aux}, \hat{M}, v, u)$ (this step requires programming H'). Since Π_{df} is $2^{-\nu+1}$-HVZK and using Fact C.8, we land in Item 1 with probability at least $\varepsilon - (Q_{\mathsf{H}}^2/2^{2\ell} + 2^{-\nu+1})$. Then, by Lemma C.3, we obtain a contradiction to strong-RSA with probability at least $(\varepsilon - Q_{\mathsf{H}}^2/2^{2\ell} - 2^{-\nu+1}) \cdot (2\phi(N)/(N-1) - 3/2)$. The claim follows. □

Therefore, with probability at least $\mathsf{p}'_{\mathsf{fork}} - \varepsilon$, we must land in Item 2 or in Item 3, and it must hold that FailSim happens and $N = p_1 p_2$ where p_1, p_2 are primes larger than $2^{n/2-\ell-\nu}$. However, the latter directly contradicts Item 2, hence it necessarily holds that E returns $(x', r) \leftarrow \mathsf{E}'(\mathsf{so}_2, \mathsf{so}'_2)$ such that

1. $(x', r) \in \pm 2^{n_x+\nu+1} \times \mathbb{Z}_N$,
2. $X = g^{x'} \in \mathbb{G}$ and $\mathcal{E} = (1+N)^{x'} r^N \bmod N^2$

Then, by Lemma D.1, this implies that $\mathsf{dec}_{\varphi(N)}(\mathcal{E}') = x' \bmod q$. However, this contradicts FailSim. This probability must therefore be 0, and we get $\mathsf{p}'_{\mathsf{fork}} = \varepsilon$. Hence,

$$\frac{p^2}{Q} - \frac{p}{2^{2\ell}} \leq \varepsilon,$$

with $p \geq \delta^2/Q - \delta/2^{2\ell} - \mathsf{Adv}^{\mathsf{dlog}}_{\mathsf{F}_{\mathcal{B}}}(1^{n+2\ell+2\nu+3} - Q/2^\ell$ and $\varepsilon \leq (\mathsf{Adv}^{\mathsf{sei}}_{\mathsf{F}_C}(1^\ell) \cdot (2\phi(N)/(N-1) - 3/2)^{-1} + 2^{-\nu+1} + Q_{\mathsf{H}}^2/2^{2\ell}$. This concludes the proof. □

D.2 Corrupted verifier

We now consider the case of a corrupted verifier. We construct a simulator Sim that is given programmable access to H' and emulates the role of a honest prover.

On input $(\mathsf{vk}^\star_{\mathsf{V}}, \mathtt{val}, X)$ with $X = \mathsf{Eval}^{\mathsf{H}}_2(\mathsf{sk}, \mathsf{vk}^\star_{\mathsf{V}}, \mathtt{val})$,

Proof. Let Q denote a bound on the number of queries of $\mathcal{A}$ to H. Sim does the following:

1. Upon receiving $\mathsf{vk}^\star_{\mathsf{V}}$ from the verifier, it parses $\mathsf{vk}^\star_{\mathsf{V}} = (\hat{M}, v, u, \pi_{\mathsf{df}})$, checks $\hat{M} \geq 2^{n_{\mathsf{rsa}}}$, $v, u \in \mathbb{Z}^*_{\hat{M}}$ and $\mathsf{vrfy}_{\Pi^{\mathsf{FS}}_{\mathsf{df}}}($, $\mathsf{aux}, \hat{M}, v, u, \pi_{\mathsf{df}}) = 1$. If verification succeeds, it sends init to the eVRF functionality.
2. It waits for the eVRF functionality to return $(\mathtt{val}, X)$ with $X \neq \perp$.
3. It samples $\sigma \leftarrow \pm\lfloor N/q \rfloor$, $\mu \leftarrow \pm 2^{n_\mu}$, and $\widetilde{P} \leftarrow v^\mu \bmod \hat{M}$.
4. It sets $\mathcal{E}' = \mathsf{H}(\mathtt{val})$ and $\mathcal{E} \leftarrow \mathcal{E}' \cdot (1 - \sigma N) \bmod N^2$.
5. It computes $\psi_{\mathsf{dlenc}} \leftarrow \mathsf{Sim}^{\mathsf{FS}}_{\mathsf{dlenc}}($, $ssid, \mathsf{P}, \mathbb{G}, g, q, X, \mathcal{E}, N, \rho, n_x, n_\lambda, \hat{M}, v, u, \widetilde{P})$ (this step requires programming H').
6. It sets $\pi = (\sigma, \psi_{\mathsf{dlenc}})$ and sends (X, π) to the verifier.

We prove that Sim's behavior is indistinguishable from that of a honest prover via a sequence of games.

Game_0. This game is the real protocol, where Sim honestly plays the role of the prover and does not interact with the eVRF functionality.

Game_1. In this game, we modify the construction of ψ_{dlenc} as follows: σ and $\widetilde{P}$ are constructed as before ($\widetilde{P} \leftarrow u^x v^\mu$) but the proof $(A, \widetilde{B}, \mathcal{D}, z_1, z_2, z_3)$ is simulated using $\mathsf{Sim}^{\mathsf{FS}}_{\mathsf{dlenc}}$, as in steps 4 and 5 of Sim's description. Using Claim 5.2 and Fact C.8, the advantage of $\mathcal{A}$ in distinguishing Game_0 from Game_1 is at most $\mathsf{Adv}^{\mathsf{sei}}_{\mathcal{A}}(1^\ell) + 2^{-\nu+2} + Q^2/2^{2\ell}$.

Game_2. In this game, we modify the way $\widetilde{P}$ is constructed: instead of setting $\widetilde{P} \leftarrow u^x v^\mu$, the simulator samples $\widetilde{P} \leftarrow\!\!\leftarrow v^\mu$, as in step 3 of Sim's description (note that σ is still constructed as before).

By the ℓ-soundness of $\Pi_{\mathsf{df}*}$, for every Q-query adversary $\mathcal{A}$, it holds with probability at least $1 - Q/2^\ell$ that there exists $\lambda \in \pm 2^{n_\lambda}$ (with $n_\lambda = 2\ell + \nu$) such that $u = v^\lambda$. If this is the case, we have $\widetilde{P} = (v^\lambda)^x \cdot v^\mu = v^{\lambda x + \mu}$. Since $\lambda x \leq 2^{4\ell+\nu}$ and μ is sampled from $\pm 2^{n_\mu}$ with $n_\mu = 4\ell + 2\nu$, the distribution of $\lambda x + \mu$ is $2^{-\nu}$-statistically close to that of μ. Therefore, the advantage of any Q-query adversary $\mathcal{A}$ in distinguishing Game_1 from Game_2 is at most $Q/2 - \ell + 1/2^\nu$.

Game_3. In this game, we samples $\sigma \leftarrow \pm\lfloor N/q \rfloor$, as in step 3 of Sim's description. Observe that in this game, Sim does not need to know x anymore, except for computing $X = g^x$. We prove the following:

Claim D.4. *The advantage of $\mathcal{A}$ in distinguishing* Game_2 *from* Game_3 *is at most* $2 \cdot \mathsf{Adv}^{\mathsf{dcr}}_{\mathcal{A}}(1^\ell) + 2 \cdot Q^2/2^{2\ell} + 2^{-\nu+1}$.

Proof. We consider a game between a challenger for the "left-or-right" IND-CPA security game of Paillier, where the adversary can adaptively send pairs (m_0, m_1) to the challenger, who replies either always with $\mathsf{enc}_N(m_0)$ or always with $\mathsf{enc}_N(m_1)$. It is a standard result that the advantage of any PPT adversary in this game tightly reduces to DCR up to a factor 2 loss. The reduction works as follows:

(1) It receives N from the challenger of the IND-CPA security game for Paillier (Definition C.6), and simulates $\mathsf{vk_P} = (N, \rho, \rho_0, \pi_{\mathsf{mod}}, \pi_{\mathsf{fac}})$ by constructing (ρ, ρ_0) honestly, simulating $\pi_{\mathsf{mod}} \leftarrow \mathsf{Sim}^{\mathrm{FS}}_{\mathsf{mod}}(\mathsf{aux}, N)$, and simulating $\pi_{\mathsf{fac}} \leftarrow \mathsf{Sim}^{\mathrm{FS}}_{\mathsf{fac}}(\mathsf{aux}, N)$ (this step requires programming H').
(2) It receives $\mathsf{vk}^\star_{\mathsf{V}}$ from $\mathcal{A}^{\mathsf{H}}(\mathsf{vk_P})$
(3) Upon receiving a query `val` from $\mathcal{A}$ to H, it samples $\mathsf{tmp}, r \leftarrow \mathbb{Z}_N$ sends (tmp, r) to the IND-CPA challenger and receives $\mathcal{E}'$ (encrypting either $\mathsf{tmp} = x + q\sigma$ or r).
(4) It programs $\mathsf{H}(x) = \mathcal{E}'$. Note that in this step, the *reduction* programs H (but Sim does not).
(5) Upon receiving a query `val` from $\mathcal{A}$ to O,
   - If `val` was previously queried to H, it retrieves tmp such that $\mathsf{H}(x) = \mathsf{enc}_N(\mathsf{tmp})$ and compute $x = [\mathsf{tmp}]_q$, $\sigma = (\mathsf{tmp} - x)/q$.
   - Else, it samples $x \leftarrow \mathbb{Z}_q$, $\sigma \leftarrow \pm\lfloor N/q \rfloor$, sets $\mathsf{tmp} = x + \sigma q$, and $X = g^x$.
   - It samples $r \leftarrow \mathbb{Z}_N$ and sends (tmp, r) to the IND-CPA challenger, receives $\mathcal{E}'$ (encrypting either $\mathsf{tmp} = x + q\sigma$ or r), and programs $\mathsf{H}(x) = \mathcal{E}'$.
   - It simulates $\widetilde{P}$ and the rest of ψ_{dlenc} as in Game_2 and sets $\pi = (\sigma, \psi_{\mathsf{dlenc}})$.
   - It returns (X, π).

   Eventually, when $\mathcal{A}$ returns a bit b, it sends b to the challenger.

It is straightforward to observe that when the challenger always encrypts the left input tmp, this game is perfectly identical to Game_2 from the viewpoint of $\mathcal{A}$: all calls to H return random elements of $\mathbb{Z}^*_{N^2}$ (as Paillier ciphertexts are uniformly distributed in $\mathbb{Z}^*_{N^2}$) and calls to $\mathsf{O}(\mathtt{val})$ return $(X, \sigma, \psi_{\mathsf{dlenc}})$ where ψ_{dlenc} is simulated, $\mathsf{H}(\mathtt{val}) = \mathcal{E}'$ is a uniform element of $\mathbb{Z}^*_{N^2}$ and denoting $\mathsf{tmp} = \mathsf{dec}_{\phi(N)}(\mathcal{E}')$, it holds that $X = g^{[\mathsf{tmp}]_q}$ and $\sigma = (\mathsf{tmp} - [\mathsf{tmp}]_q)/q$. Similarly, when the challenger always encrypts the right input r, this game is perfectly identical to Game_3 (where $\mathcal{E}'$ is random and independent of X, and σ is random and independent of $\mathcal{E}'$). It follows that the reduction wins with the same probability as $\mathcal{A}$. The claim follows. □

Game_4. In this last game, Sim does not compute x as $[\mathsf{dec}_{\phi(N)}(\mathcal{E}')]_q$ anymore. Instead, Sim behaves exactly according to step 1 to step 6: it waits for the eVRF functionality to output X and behaves as in Game_3 to emulate $\pi = (\sigma, \psi_{\mathsf{dlenc}})$. It outputs (X, π). Note that from Game_3, π is independent of X: the only difference between Game_3 and Game_4 is that, denoting $X = g^x$, x is equal to $[\mathsf{dec}_{\phi(N)}(\mathcal{E}')]_q$ in Game_3, while x is a uniformly random $\mathbb{Z}_q$ element in Game_4 (as it is sampled by the eVRF functionality). Hence, the indistinguishability of Game_3 and Game_4 follows immediatly from the fact that the mapping $(x, \phi(N)) \mapsto [\mathsf{dec}_{\phi(N)}(\mathsf{H}(x))]_q$ is a PRF with range $\mathbb{Z}_q$. In turns, this reduces to the DCR assumption in the random oracle model via a straightforward reduction. This concludes the proof. □

E POC SCREENSHOTS

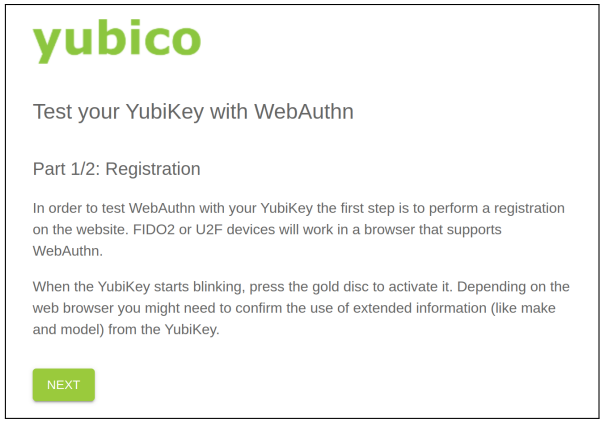

Figure 2: Yubico's Server Registration Init

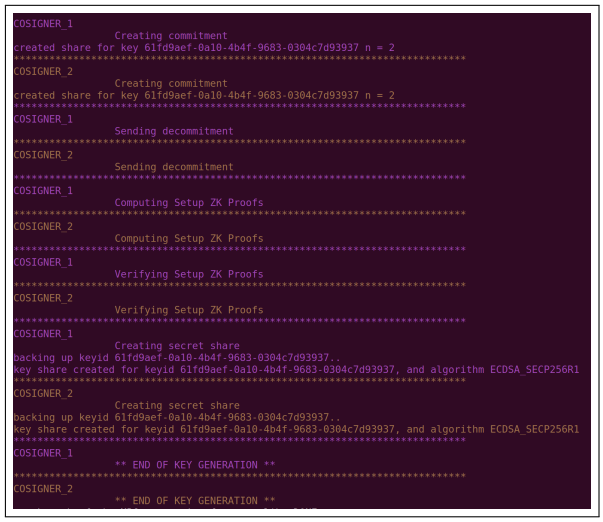

Figure 3: MPC KeyGen

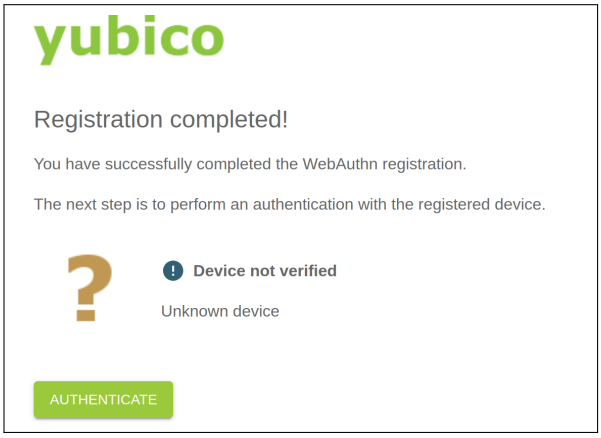

Figure 4: Yubico's Server Registration Successful

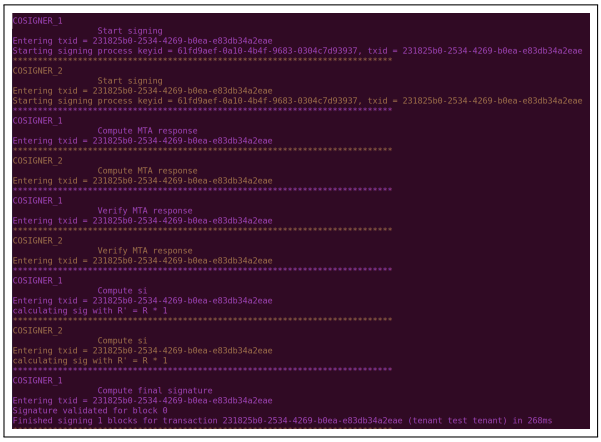

Figure 5: MPC Signature

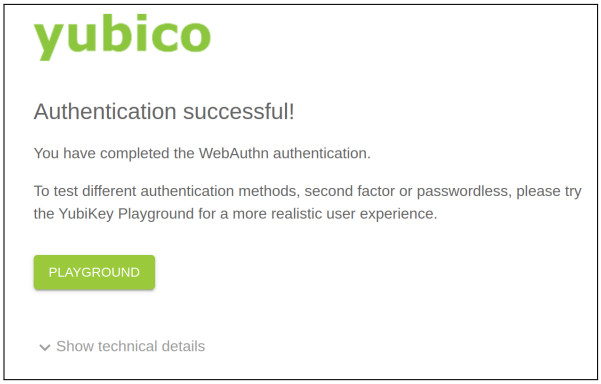

Figure 6: Yubico's server Authentication