Probabilistic Atomic Swaps for Bitcoin and Friends

Paul Gerhart[1], Jay Taylor[2], and Sri Aravinda Krishnan Thyagarajan[2]

[1] TU Wien, Austria
[2] University of Sydney, Australia

Abstract. Atomic swaps are a fundamental primitive for the trustless exchange of digital assets across blockchains: they guarantee that either both parties receive the agreed assets or neither party transfers. While this all-or-nothing guarantee is powerful, it also imposes an inherent determinism that rules out exchanges whose intended outcome is probabilistic. As a result, existing atomic swaps cannot realize trustless exchanges in which one party pays for a fixed chance of receiving a larger asset or reward, as in lotteries, randomized allocation mechanisms, and probabilistic cross-chain trades.

We introduce *probabilistic swaps*, a new cryptographic primitive that extends atomic swaps to the probabilistic setting. In a probabilistic swap, one party's transfer is executed with a fixed, publicly specified probability embedded in the protocol and cannot be biased by either party. This yields a trustless mechanism for randomized exchange with verifiable odds and no trusted intermediary.

Our construction combines adaptor signatures with oblivious pseudorandom functions (OPRFs) to realize the desired probabilistic outcome while ensuring that neither party can predict or bias it in advance. Along the way, we introduce a new mechanism for the atomic exchange of OPRF evaluations for payments, which may be of independent interest. A key feature of our approach is that it preserves the minimal on-chain footprint of modern atomic-swap protocols. The protocol relies only on standard Bitcoin scripts, such as digital signatures and timelocks, and is deployable on any blockchain that already supports atomic swaps. Consequently, probabilistic swaps are indistinguishable from ordinary on-chain transactions, which helps preserve privacy and fungibility. We provide formal security foundations and demonstrate practicality through a probabilistic swap between the Bitcoin and Litecoin testnets, as well as in the Lightning Network.

Keywords: Probabilistic Atomic Swap, Adaptor Signature, Oblivious PRF, OPRF

1 Introduction

Atomic swaps of digital assets are a fundamental economic primitive: they allow two parties to exchange assets with the guarantee that either both receive their desired assets or neither does [50]. One of the key motivations behind blockchain systems is precisely to enable such exchanges without a trusted intermediary, and this has attracted significant interest from both academia and industry [35,55,29,52,61,34,4], with practical relevance for cross-chain interoperability and decentralized finance [47,49,61,46,29,52,58].

Classical atomic swaps enforce a deterministic exchange relation: if one party obtains the counterparty's asset, then the counterparty obtains the agreed asset as well. This all-or-nothing guarantee is powerful, but it cannot directly express exchanges whose intended output is a probability distribution rather than a fixed transfer. In this work, we ask whether one can realize a *probabilistic swap*: a contract-minimal exchange in which one party pays for a publicly specified chance of receiving an asset, while the protocol guarantees that the realized outcome has exactly the advertised distribution and cannot be biased by either party.

One motivating example is *wagers*. In such a setting, a participant commits a fixed amount in exchange for a chance to receive a larger reward (from another party) according to predefined odds. This application is also popularly referenced as `OP_RAND` in the Bitcoin community [43], and has been identified as a promising tool for handling tiny, otherwise "uneconomical" payments on the Lightning Network [1]. Our work realizes exactly this vision, as we demonstrate in Section 7.

Another natural example is *probabilistic cross-chain exchange*. A user holding a long-tail asset may prefer a small, explicitly priced chance of receiving a highly liquid asset, such as Bitcoin, over a guaranteed but

unfavorable exchange rate. A probabilistic swap lets the parties encode this trade directly: the user transfers the source asset in exchange for a fixed probability of receiving the target asset, with the odds and payoffs agreed in advance.

The key requirement in both these examples is that, once the payment is made, the outcome is realized with exactly the *advertised* probability, and neither party can bias or influence the result. Unlike traditional scenarios that rely on a trusted house or intermediary, the setting here is decentralized, and the parties obtain cryptographic guarantees about the fairness of the outcome and the correctness of the winning probability.

Existing work. Prior approaches have explored blockchain-based randomized protocols, including lottery-style mechanisms on Bitcoin, showing that probabilistic outcomes can in principle be realized in a trustless setting [42,48]. However, these constructions typically rely on several assumptions. Firstly, they require complex smart contracts or intricate script-level mechanisms, such as carefully structured spending conditions, to enforce correctness [48,3,10]. This complexity has concrete costs: larger transactions and higher fees, more distinguishable on-chain behavior, and reduced portability to chains with limited scripting support. This complexity comes at huge costs: (1) larger transactions and higher fees, (2) more distinguishable on-chain behavior, weakening on-chain privacy and fungibility, and (3) limited compatibility across blockchains with limited scripting/contract support. These downsides of excessive script-usage have already been highlighted in several prior works [60,5,61,34,63] in the context of other payment applications. Secondly, existing solutions like [48,8,3] for lotteries require the set of participants to be fixed and known to everyone ahead of time, which makes them poorly suited to open, permissionless participation. As a result, while such approaches demonstrate theoretical feasibility, they do not provide a rigorous abstraction for decentralised probabilistic exchange analogous to atomic swaps. We therefore ask the question:

Can we extend the atomic swap of blockchain tokens into the probabilistic setting without modifying the existing minimal on-chain footprint?

We emphasize that this question is not merely of theoretical interest as it addresses a fundamental limitation of classical atomic swaps and opens up new benefits for applications with randomized outcomes on blockchain systems like Bitcoin.

A new primitive. We answer this question affirmatively by introducing *probabilistic swaps*, a cryptographic primitive that extends atomic swaps from deterministic exchanges to exchanges with a prescribed outcome distribution. In a probabilistic swap between a dealer D and a party P, P transfers an asset in exchange for a fixed probability p of receiving D's asset; with probability $1-p$, P receives nothing. The probability p is fixed in advance and enforced by the protocol: after the parties commit, neither party can bias the outcome, selectively abort after learning it, or obtain value outside the prescribed distribution. If the swap does not proceed, then neither side's asset is transferred. Thus, probabilistic swaps retain the trustless-exchange guarantee of atomic swaps while replacing deterministic delivery with fair probabilistic delivery. We give a high-level overview in Figure 1.

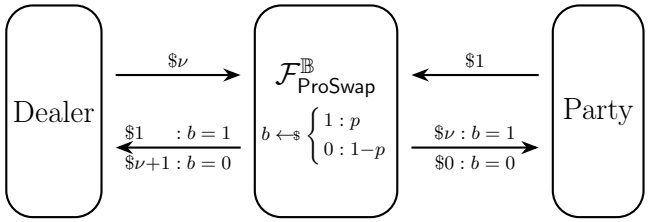


Fig. 1. Ideal functionality $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$ for probabilistic swaps. The dealer inputs ν coins and the party inputs one coin. The functionality samples $b \leftarrow\!\$$ Bernoulli(p) and always transfers one coin to the dealer. If $b=1$, the party receives ν coins; otherwise, the dealer is refunded its ν coins. Probabilistic swaps subsume atomic swaps [61] when setting $p=1=\nu$.

A key advantage of our approach is its minimal reliance on blockchain functionality. Our protocol can be implemented on *any blockchain* that supports atomic swaps [61], *without any* additional on-chain overhead and requiring only lightweight off-chain cryptographic techniques. This makes probabilistic swaps immediately deployable on a wide range of existing systems, including script-limited and cross-chain environments.

1.1 Our contributions

Our main contributions can be summarised as follows:

- We introduce *probabilistic atomic swaps*, a new cryptographic primitive that extends classical atomic swaps to probabilistic settings. We formalize this notion via an ideal functionality (c.f., Section 4) that captures fair exchange with an a priori specified probability p. Our new functionality is a generalization of the regular atomic swap functionality [61] simply by setting $p = 1$. Importantly, our formulation preserves the lightweight nature of atomic swaps by ensuring that probabilistic swaps incur the same on-chain footprint as their deployed deterministic counterparts.
- We construct an efficient protocol for probabilistic atomic swaps using only minimal script support: transaction signature verification and timeouts. These features are available on Bitcoin-like chains and are compatible with payment-channel frameworks. We give a concrete instantiation from Schnorr signatures, and more generally identify the compatibility requirements needed for other adaptor signature relations, and cryptographic tools like encryption keys, oblivious pseudorandom function (OPRF) output spaces, and well-formedness proofs [4,56,28,27]. Our techniques can also be extended to chains without native transaction timeouts using cryptographic timed-payment methods [61].
- We develop, as a central technical ingredient and contribution of independent interest, a mechanism for *atomic exchange of oblivious PRF evaluation for payments*. An OPRF [24] allows a client to obtain the evaluation of a PRF on an input using a server's secret key, without the server learning the input and without the client learning anything about the key beyond the output. In this mechanism, the dealer acts as an OPRF server, and the party obtains OPRF evaluations if and only if it makes the required payment. Beyond enabling probabilistic atomic swaps, this tool may be broadly useful in OPRF-based applications, including payment-driven rate limiting in password authentication, or encrypted backup systems [15,21,39,2,36,17]. In particular, we envision payment-based rate limiting to replace hard cut-offs after a fixed number of attempts with a small fee per authentication attempt. This imposes negligible cost on honest users, especially since our protocol can be instantiated entirely on layer-2 networks, thereby minimizing fee overhead. At the same time, it renders large-scale brute-force attacks economically infeasible. We leave the exploration of this idea as interesting future work.
- We implement and evaluate our protocol on the Bitcoin and Litecoin testnets, demonstrating practical feasibility and low on-chain overhead. A complete on-chain swap uses exactly four standard Taproot transactions; in our testnet experiments, these transactions were about 575WU each. We also demonstrate a Bitcoin–Litecoin cross-chain swap. For layer-2 deployment, PTLC-style systems support the adaptor signature version of our protocol; since the deployed Lightning Network is currently *hash timelock contract* (HTLC)-based, our artifact also includes an HTLC/preimage variant with explicit hash/discrete-log link proofs. This provides a concrete realization of an `OP_RAND`-style functionality for Lightning-like payment networks [1].

1.2 Related Work

Prior work demonstrates that probabilistic exchanges can be realized on blockchains, for example, in decentralized poker [42] and lottery protocols [48,8], or general MPC [3,10]. However, these protocols either require additional opcodes not supported by Bitcoin [42,8], achieve guarantees only via security deposits as least as big as all coins to be exchanged [3,10], or rely on advanced scripting capabilities that publicly reveal both the execution of a lottery and its conditions [48,8]. We compare these approaches in Table 1.

(Deterministic) Atomic swaps enable trustless exchange across blockchains. Early constructions such as TumbleBit [35] and later protocols including A^2L [55,29] and BlindHub [52] focus on practical, privacy-preserving implementations. More recent works, such as Universal Swaps [61] and Sweep UC [34], provide

more general and formally secure frameworks. Atomic swaps are also used to build payment channels [4]. Our formalization builds upon the security framework of [61] but is enhanced to support probabilistic exchanges. Along the way, we identified minor formalization gaps in the functionality of [61], in particular a mismatch between the ideal functionality and real-world behavior: while the ideal functionality refunds both parties atomically, an adversary in the real world may refuse to reclaim locked funds, leading to executions that cannot be simulated. We refer to Section 4 for more discussion.

Table 1. Comparison of on-chain protocols. With private protocol semantics, we denote whether the protocol execution can be inferred from on-chain data.

	This work	**OP_Rand** [43]	**Lottery** [48,8]	**UnivSwps** [61]	**Poker** [42]	**MPC** [3,10]
Probabilistic output distribution	✓	✓	✓	✗	✓	✓
On-Chain privacy	✓	✓	✗	✓	✗	✗
Off-chain/cross-chain compatible	✓	✓	✗	✓	✗	✗
As-of-today Bitcoin compatible	✓	✓	✓	✓	✗	✓
On-chain cost: moderate (→)/ low (↓)	↓	↓	→	↓	→	→
Deposit-based (no deposits are better)	✗	✗	✗	✗	✓	✓
Proven secure	✓	✗	✓	✓	✓	✓

Most directly related to our protocol are universal swaps [61] and the Bitcoin lotteries of [48,8]. Our protocol, as well as the lotteries, supports probabilistic exchanges, whereas universal swaps focus on deterministic exchanges. On-chain, our protocol behaves similarly to universal swaps, revealing only freezing and exchange transactions without exposing additional protocol-specific information. This results in an on-chain footprint of four standard transactions: two that each transfer coins from a party's address to a temporary address to freeze them. And two transactions from the temporary address to either party, (or both to the dealer, if the party lost). In contrast, the Bitcoin lottery protocols [48,8] require encoding the full protocol logic on-chain, including the outcome probabilities. Consequently, on-chain costs involve evaluating complex scripts, whereas our protocol relies solely on standard transaction signatures and timelock verification. The overhead of timelocks can be further reduced by instantiating our protocol with verifiable timed signatures (VTS) [59], similar to [61]. Finally, since the atomicity of the exchange in our protocol does not rely on expressive on-chain scripting, it can be integrated into existing exchange frameworks, such as layer2 networks, and naturally supports cross-chain execution. This is not possible for the lottery protocol [48].

Independently in his note, Kurbatov [43] proposes a protocol for emulating an `OP_RAND` opcode on Bitcoin via a two-party interactive protocol. The overall structure is similar to ours: a party makes a guess, and if correct, learns a witness that allows it to spend coins. However, the two works differ substantially in two ways. First, in [43] *both* the dealer and the party must compute NIZKs over hash evaluations, with the dealer's proof growing linearly in m: for winning probability $1/m$, the dealer proves $m+1$ hash instances including a `hash160` evaluation, while the party proves one `hash160` evaluation. In contrast, our protocol requires the dealer only to compute a single well-formedness proof, while the party computes no NIZK at all; our Bulletproof instantiation achieves near-constant proving time regardless of m. Therefore, for probabilities other than $1/2$, the approach of [43] quickly becomes impractical, whereas our protocol remains efficient even for probabilities as small as 2^{-20}. This matters in practice: encoding a one-satoshi payment on the Lightning Network as a probabilistic payment requires winning probabilities on the order of 2^{-17} or smaller, a regime where [43] becomes prohibitively expensive. Second, [43] provides no formal security definitions or proofs. Therefore, while the protocol appears intuitively sound, it remains unclear whether it achieves the atomicity and bias-resistance guarantees we formalize in Section 4.

2 Preliminaries

We denote by λ the security parameter. We say an algorithm is efficient if it can be computed in probabilistic polynomial time (PPT). We say a function $\mu(\cdot)$ is negligible if it decreases faster than the inverse of any polynomial. By $\mathcal{X} \approx_c \mathcal{Y}$, we denote computational indistinguishability of the two distributions $\mathcal{X}$ and $\mathcal{Y}$. We defer formal treatments of OPRFs, adaptor signatures, and NIZKS to Appendix A.

Security definition. We use a standard simulation-based notion of security [44]. Let Π be a protocol, $\mathcal{F}$ be an ideal functionality, $\mathcal{A}$ be a probabilistic polynomial-time adversary, and Sim be a probabilistic polynomial-time simulator. We define the random variables $\texttt{IDEAL}_{\mathcal{F},\mathsf{Sim}}$ consisting of the identity of the corrupted party, the outputs of the honest parties, and the output of Sim, and $\texttt{REAL}_{\Pi,\mathcal{A}}$ consisting of the identity of the corrupted party, the outputs of the honest parties, and the view of $\mathcal{A}$ at the end of the execution of Π.

Digital signatures. A digital signature scheme Σ consists of three algorithms: a key generation algorithm $\mathsf{KGen}(\lambda)$ that reads a security parameter and outputs a key-pair $(\mathsf{pk}, \mathsf{sk})$, a signing algorithm $\mathsf{Sign}(\mathsf{sk}, m)$ that reads the signing key sk, and a message m and outputs a signature σ, and a verification algorithm $\mathsf{Vrfy}(\mathsf{pk}, m, \sigma)$ that reads the public key pk, a message m, and signature σ and outputs 1 if σ is valid and 0 otherwise. We require Σ to be existentially unforgeable under chosen-message attacks (EUF-CMA) [41].

Hard relations A relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ is associated with the language $\mathcal{L}_\mathcal{R} := \{\mathbb{x} \mid \exists \mathbb{w} : (\mathbb{x}, \mathbb{w}) \in \mathcal{R}\}$. We say that $\mathcal{R}$ is a hard relation [4] if: (i) there exists a PPT algorithm $\mathsf{genR}(\lambda)$ that outputs $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, (ii) membership in $\mathcal{R}$ can be decided in PPT, and (iii) for every PPT adversary $\mathcal{A}$, given $\mathbb{x}$, it is infeasible to compute a corresponding witness $\mathbb{w}$.

Adaptor signatures. Let $\mathcal{R}$ be a hard relation and let Σ be a signature scheme. An adaptor signature scheme is a tuple $\mathsf{AS} = (\mathsf{pSign}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract})$, where $\mathsf{pSign}(\mathsf{sk}, m, Y)$ outputs a pre-signature $\widetilde{\sigma}$, $\mathsf{pVrfy}(\mathsf{pk}, m, Y, \widetilde{\sigma})$ verifies its validity, $\mathsf{Adapt}(\widetilde{\sigma}, y)$ outputs a signature σ, and $\mathsf{Extract}(\widetilde{\sigma}, \sigma)$ outputs a witness y.

We require AS to be *adaptable*, ensuring that any valid pre-signature can be adapted into a valid signature given a valid witness, and *extractable*, guaranteeing that any valid signature was either issued by the signer or enables extraction of a valid witness with respect to a pre-signature issued by the signer [28].

Oblivious PRFs. An oblivious pseudorandom function (OPRF) [24] allows a client to obtain an evaluation of a PRF on input x under a server-held key sk, without revealing x to the server and without learning sk. OPRFs consist of four algorithms $(\mathsf{Req}, \mathsf{BlindEval}, \mathsf{Finalize}, \mathsf{Eval})$. We instantiate OPRFs using the 2-Hash-DH construction [37]. Let $\mathbb{G}$ be a cyclic group of prime order p, and let H_p and $\mathsf{H}_\mathbb{G}$ be hash functions mapping to $\mathbb{Z}_p$ and $\mathbb{G}$, respectively. To form a request for input x, the client runs $\mathsf{Req}(x)$ by sampling $r \leftarrow\!\$\ \mathbb{Z}_p$ and computing $\mathsf{req} = \mathsf{H}_\mathbb{G}(x)^r$. Upon receiving req, the server computes $\mathsf{res} = \mathsf{BlindEval}(\mathsf{sk}, \mathsf{req}) = \mathsf{req}^{\mathsf{sk}}$. The client then runs $\mathsf{Finalize}(x, r, \mathsf{res})$ to remove the blinding and obtain $\mathsf{Eval}(\mathsf{sk}, x) = \mathsf{H}_p(x, \mathsf{res}^{1/r}) = \mathsf{H}_p(x, \mathsf{H}_\mathbb{G}(x)^{\mathsf{sk}})$.

NIZKs A non-interactive argument system (NARG) for a relation $\mathcal{R}$ in the random oracle model is a tuple $\Pi = (\mathcal{P}, \mathcal{V})$, where $\mathsf{Setup}(\lambda)$ outputs public parameters, $\mathcal{P}^\mathsf{H}(\mathbb{x}, \mathbb{w})$ outputs a proof π for $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, and $\mathcal{V}^\mathsf{H}(\mathbb{x}, \pi)$ outputs 1 or 0. We require completeness, i.e., valid statements admit accepting proofs, and soundness, i.e., no efficient adversary can produce a proof for a false statement. The system is zero-knowledge if there exists a simulator that can generate proofs for true statements without access to the witness such that no efficient adversary can distinguish simulated proofs from real ones. A NARG with statistical soundness is called a non-interactive zero-knowledge proof (NIZK).

Communication model. We consider a synchronous communication model in which parties interact in rounds, and messages sent in a round are delivered within that same round. We assume static corruptions, i.e., the adversary chooses the corrupted party before the protocol execution and cannot change it thereafter. This model is standard in the context of atomic swap protocols [61].

Blockchain. We follow the blockchain model of [61] and assume an ideal ledger functionality $\mathcal{F}_{\mathbb{B}}$ that we model as a trusted append-only bulletin board maintaining the balances associated with each address. Parties can register addresses and transfer coins via a $\mathsf{Post}(A, B, v)$ interface, which moves v coins from address A to B, and can query the current state of the ledger via a Read interface. Following [4,61], the ideal blockchain functionality accepts any posted transaction without enforcing authorization by a specific identity. In a real-world instantiation, however, transactions must be authorized via digital signatures. We address this gap by having the simulator forward Post requests only upon receiving a valid signature on the transaction, and by including these signatures in the adversary's view upon Read queries.

Timelocks. We use absolute timelocks as specified in BIP 65 via `OP_CHECKLOCKTIMEVERIFY` (CLTV) using two spending branches. One branch allows spending by $\mathsf{pk}_{\mathsf{tmp}}$ at any time. The other branch allows spending by pk, provided that time $\mathbf{T}$ has been reached. For convenience, we denote this script by $\Lambda(\mathsf{pk}_{\mathsf{tmp}}, \mathbf{T}, \mathsf{pk})$, which expands to the locking script shown in Figure 2.Instead of using timelocks, our protocol could also be realized using verifiable timed payments as shown in [61] (c.f., Section 6).

```
OP_IF pk_tmp OP_CHECKSIG
OP_ELSE
  T OP_CHECKLOCKTIMEVERIFY OP_DROP
  pk OP_CHECKSIG
OP_ENDIF
```

Fig. 2. Transaction output using BIP 65. This output can be spent by $\mathsf{pk}_{\mathsf{tmp}}$ at any time, and by pk after time $\mathbf{T}$.

3 Solution Overview

Before presenting our solution for probabilistic swap protocols, let us first recall our setting. We consider two entities, a dealer and a party, who wish to engage in a coin exchange with asymmetric and probabilistic outcomes. In contrast to deterministic atomic swaps, the dealer receives a fixed, typically smaller, payment of 1 coin, while the party receives a higher reward, say ν_{D} coins, only with some probability $p \in [0, 1]$. The outcome is determined by a probabilistic event agreed upon in advance by both parties and cannot be influenced by either party.

Deterministic atomic swaps. We begin by recalling the deterministic atomic swaps on scriptless blockchains enabled by adaptor signatures from [61]. On a high level, in a deterministic atomic swap, the dealer samples a statement-witness pair (Y, y) of a hard relation and sends the statement Y to the party together with an adaptor signature *pre-signature* $\widetilde{\sigma}_{\mathsf{DP}}$ with respect to Y. This pre-signature corresponds to a transaction transferring coins from the dealer to the party. Upon receiving the pre-signature, the party constructs a different transaction transferring coins from the party to the dealer, produces a corresponding pre-signature $\widetilde{\sigma}_{\mathsf{PD}}$ with respect to the same statement Y, and sends it to the dealer. If the dealer decides to execute the swap, it uses its knowledge of a valid witness y to adapt the party's pre-signature $\widetilde{\sigma}_{\mathsf{PD}}$ into a valid signature σ_{PD} (using the interface from adaptor signatures), broadcasts the party-to-dealer payment transaction on-chain, and can then go offline. The party eventually observes the adapted signature on-chain and extracts the witness y using the extractability property of adaptor signatures. Knowing y, the party can, in turn, adapt the pre-signature $\widetilde{\sigma}_{\mathsf{DP}}$ received from the dealer on the dealer-to-payment transaction and claim its payment.

This exchange is atomic for the dealer, since it reveals the witness y only by completing the transaction that pays it. If the dealer does not proceed, the party cannot obtain y and therefore cannot claim the dealer's coins. It is also atomic for the party, since whenever the dealer completes the transaction and receives payment, the party can extract y from the on-chain signature and use it to claim its own payment. We note

that such a protocol requires temporarily locking coins to prevent frontrunning attacks. For simplicity of exposition, we omit these locking mechanisms in this overview.

3.1 A Strawman Approach

On a high level, deterministic atomic swaps have a similar setting to the probabilistic version we seek. Therefore, as a natural starting point, we try to modify that paradigm to obtain probabilistic behavior. A first idea is to make *adapt* probabilistic: upon learning a valid witness y, the party can adapt its pre-signature only with probability p. The parties would then exchange pre-signatures exactly as in the deterministic protocol. The dealer would still obtain payment first, while the party would only be able to complete its side of the swap, and hence receive payment, with probability p. Similar in flavor, but using another interface of adaptor signatures, one could also think about making witness extraction probabilistic.

However, neither of these approaches achieves the desired notion of probabilistic exchange. The core problem is that once the dealer receives the party's pre-signature, the dealer essentially has the same information as the party for evaluating whether the party will ultimately succeed. In particular, the dealer can run the same adapt or extraction procedure locally and determine whether the party can claim the reward. Thus, the dealer can predict the protocol outcome before the exchange is completed and abort whenever the outcome is unfavorable. For example, in a wager-style execution, the dealer could learn in advance that the party is going to win and simply refuse to proceed.

This is not merely an artifact of a particular adaptor signature construction. Rather, it reflects a more basic obstacle: in a probabilistic swap, the outcome must remain hidden from both parties until they are already committed to the exchange. Otherwise, the party that learns the outcome early can selectively abort, thereby biasing the protocol's effective outcome. We return to this broader issue in Section 6, where we also discuss the difficulty of realizing probabilistic swaps in which both parties receive probabilistic outcomes on minimal blockchains.

3.2 Towards the Solution

Instead of making adapt or extraction itself probabilistic, we encode the uncertainty into some *hidden information* chosen by both the dealer and the party. Concretely, the dealer samples a "target value" from a sufficiently large domain, and the party succeeds only if it "guesses" this value correctly. Crucially, the party's guess must remain hidden from the dealer (similar to the dealer's choice from the party) until the dealer completes its part of the exchange irrevocably. In this way, neither party can predict the outcome in advance: the party learns whether it can complete the swap only after paying the dealer, while the dealer cannot tell ahead of time whether the party's guess was correct. By appropriately choosing the domain size for the dealer's guessing space, we can ensure that the party succeeds with probability exactly p.[3]

A First attempt. Our first attempt to realize this idea is based on oblivious transfer [20]. In an oblivious transfer (OT) protocol, a sender holds multiple messages, and a receiver obtains exactly one of them without revealing its choice to the sender, while learning nothing about the remaining messages. A natural approach is to adapt OT to our setting as follows: the dealer (the sender in OT) prepares m statement–witness pairs, pre-signs a pre-signature paying the party $\widetilde{\sigma}_{\mathsf{DP}}$ w.r.t. exactly one of the statements, while the party (the receiver in OT) uses the OT to obtain the witness corresponding to exactly one statement. If the party's choice is correct, it can adapt the dealer's pre-signature and win. By the security of OT, the party's choice remains oblivious to the dealer, so it cannot anticipate the output, which was one half of our goal.

However, the other half of our goal, namely that the party cannot anticipate the protocol outcome prior to the exchange, cannot be covered with the OT-based approach. This is the case since in OT, the party knows which element it selects. Consequently, since the party can identify in advance which of m

[3] With a single guess, the party has a winning probability of $1/m$, where m is the domain size. However, we can modify our protocol, such that it supports a concurrent guesses, resulting in a probability of a/m (c.f., Section 6).

statements corresponds to the valid adaptor signature (simply by running pre-verification with all dealer-provided statements and the check passes for exactly one statement), it can simply choose the correct witness with certainty, eliminating the intended probabilistic behavior. We could fix this gap by letting the dealer provide a uniformly random witness to the party, but this would open the door to an attack in which the dealer deliberately always sends the wrong witness to the party without being detected.

This mismatch reflects a more fundamental tension. On the one hand, the dealer must be prevented from cheating, meaning it should be guaranteed that the revealed witness corresponds to the party's actual choice. On the other hand, the party must also be prevented from cheating, which requires that it cannot determine in advance which witness is the "correct" one to learn. These two requirements are inherently at odds in OT: ensuring verifiability of the chosen message tends to leak information about correctness, while hiding choice-correctness from the party prevents enforceability of the choice. Because of this limitation, we move away from OT and instead look for other cryptographic primitives.

OPRFs for oblivious guessing. We seek a primitive, in which the dealer verifiably but obliviously evaluates the party's guess, and at the same time, the party is oblivious about the correctness of its guess until after the interaction with the dealer. Oblivious pseudorandom functions (OPRFs) [24] provide exactly this type of asymmetry. In an OPRF protocol, a server holding a secret key sk interacts with a party holding an input x. At the end of the protocol, the party learns the value $\mathsf{OPRF.Eval}(\mathsf{sk}, x)$, while the server learns nothing about x or the corresponding function value. Moreover, the party obtains the function value on only one input per execution. We use the OPRF input space as the space of possible guesses. The dealer chooses a hidden target value y_{tgt}, while the party chooses a guess y_{gss}. The party then interacts with the dealer, who acts as the OPRF server, to obtain $\mathsf{OPRF.Eval}(\mathsf{sk}_{\mathsf{D}}, y_{\mathsf{gss}})$. By OPRF security, the dealer learns neither the party's guess nor whether it was correct.

We can connect this guessing mechanism to adaptor signatures if the dealer encodes the target value into the adaptor statement. Concretely, the dealer constructs the statement Y_{win} so that there exists exactly one value y_{tgt} satisfying $Y_{\mathsf{win}} = g^{\mathsf{OPRF.Eval}(\mathsf{sk}_{\mathsf{D}}, y_{\mathsf{tgt}})}$, where g is a generator of the group used by the adaptor signature scheme. The dealer then pre-signs the payment to the party with respect to Y_{win} and sends the resulting pre-signature $\widetilde{\sigma}_{\mathsf{DP}}$ together with Y_{win}, and a proof of well-formedness of Y_{win} that we discuss later in this section. By the pseudorandomness of OPRF, the statement Y_{win} leaks no information about y_{tgt}, and the only way for the party to check if y_{gss} is correct is to evaluate the OPRF on y_{gss}. If $y_{\mathsf{gss}} = y_{\mathsf{tgt}}$, the resulting evaluation $\mathsf{OPRF.Eval}(\mathsf{sk}_{\mathsf{D}}, y_{\mathsf{gss}})$ is a valid witness corresponding to Y_{win}, and so the party can adapt the pre-signature $\widetilde{\sigma}_{\mathsf{DP}}$ and claim the payment. If the guess is incorrect, the OPRF output is useless for adapting the pre-signature. Thus, when the target is sampled uniformly from a domain of size m, a random guess of the party succeeds with probability $1/m$. This captures the probabilistic aspect of the party's payout, and we visualize this guessing strategy in Figure 3.

At this point, however, only one part of the problem has been addressed. Our OPRF-based approach safeguards the odds for the party, but they do not yet explain how to make the learning of the OPRF evaluation itself atomic with payment, i.e., make the exchange atomic for the dealer. If the dealer simply returns the OPRF response first, the party may obtain the reward (if $y_{\mathsf{gss}} = y_{\mathsf{tgt}}$) without paying. If the party pays first, the dealer may abort without delivering the response. Thus, the remaining challenge is to make *OPRF evaluation itself* the object of an atomic exchange.

The missing piece: OPRF evaluation as a service. To atomically exchange OPRF evaluations, our idea is to treat the dealer's OPRF evaluation response as *locked* until the party-to-dealer payment is completed. Rather than sending the response to the party directly, the dealer encrypts it under an ephemeral public key Z for which the dealer knows the corresponding decryption key z, and proves well-formedness of this ciphertext w.r.t. the request sent by the party (using a NIZK proof, see Appendix B). Then, the dealer sends a pre-signature $\widetilde{\sigma}_{\mathsf{DP}}$, pre-signed under the winning statement Y_{win}, alongside the ephemeral key Z, the ciphertext, and the proof of well-formedness. Upon receiving this information, the party can verify that the dealer has provided a valid response, which corresponds to evaluating the OPRF on the party's guess y_{gss}, but cannot yet recover the resulting OPRF value. This simplifies the exchange, since now it is atomic if the party can obtain the decryption key z exactly when the dealer is paid.

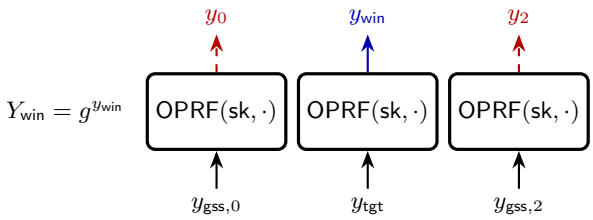


Fig. 3. The dealer acts as the OPRF server. The party submits guesses $y_{\mathsf{gss},0}, y_{\mathsf{tgt}}, y_{\mathsf{gss},2}$ and obtains the corresponding OPRF outputs. Only one guess y_{tgt} corresponds to a valid witness y_{win}; the remaining outputs (red, dashed) are invalid.

This is where adaptor signatures enter again. The party prepares the dealer's payment transaction $\widetilde{\sigma}_{\mathsf{PD}}$, but using the ephemeral key Z as the adaptor statement and sends the resulting pre-signature to the dealer. Since the dealer knows the ephemeral decryption key (which now also acts as the adaptor witness) z, it can adapt the pre-signature into a full signature σ_{PD} and publish the party-to-dealer payment transaction on-chain to claim payment. Once this signature is posted, the party can extract the adaptor witness (aka, the ephemeral decryption key) z from the adapted signature. This key z allows the party to decrypt the previously received ciphertext and recover the OPRF evaluation on y_{gss}.

The protocol ties the two events together: the dealer can obtain payment only by revealing the decryption key, and the party can obtain the OPRF evaluation only after that key is revealed through the on-chain signature. This is the mechanism that upgrades the probabilistic OPRF-based idea into an atomic probabilistic swap. We visualize this OPRF evaluation for a payment in Figure 4.

3.3 Combining the Building Blocks

At a high level, the protocol combines two ingredients: an OPRF-based guessing mechanism that introduces the desired outcome probability for the party, and an atomic OPRF-evaluation service that ensures the dealer is paid. The dealer samples a target value y_{tgt} from a domain of size m, constructs the corresponding payment statement Y_{win}, and sends the party a pre-signature $\widetilde{\sigma}_{\mathsf{DP}}$ together with the required well-formedness proofs. The party, in turn, selects a guess y_{gss} from the same domain and engages in the OPRF evaluation protocol with the dealer. This OPRF evaluation is atomically tied to a payment from the party to the dealer using the encryption-and-adaptor-signature mechanism above. If $y_{\mathsf{gss}} = y_{\mathsf{tgt}}$, which happens with probability $1/m$, the party obtains the correct OPRF evaluation, derives the witness y_{win}, and successfully adapts the pre-signature $\widetilde{\sigma}_{\mathsf{DP}}$ to claim the payment. Otherwise, the party learns nothing useful for adaptation, while the dealer is compensated for the OPRF evaluation. We illustrate the full protocol in Figure 4 and give its formal description in Section 5.

Security intuition. We briefly sketch why the proposed probabilistic swap protocol is secure and defer the formal treatment to Section 5. From the party's perspective, knowledge soundness of the NIZK proofs and correctness of the adaptor signature scheme ensure that the dealer's pre-signature $\widetilde{\sigma}_{\mathsf{DP}}$ is well-formed with respect to a target value y_{tgt} from an appropriate domain matching the expected winning probability. Hence, if the party's guess y_{gss} matches the dealer's hidden target y_{tgt}, then an OPRF evaluation yields exactly the witness y_{win}, which enables the party to adapt $\widetilde{\sigma}_{\mathsf{DP}}$ and claim the payment. Moreover, by the extractability of adaptor signatures, whenever the dealer completes the payment transaction and obtains compensation, the party can extract the decryption key z from the on-chain signature and thereby recover the OPRF output. Finally, the input privacy of the OPRF guarantees that the dealer learns no information about y_{gss}, preventing it from biasing the outcome based on the party's choice.

From the dealer's perspective, zero-knowledge of the NIZK proofs and the CPA security of the encryption scheme ensure that no information about the target value y_{tgt} or the corresponding OPRF evaluation is revealed prior to payment from the party. At the same time, the adaptor signature mechanism guarantees

that the ephemeral decryption key is disclosed to the party only if the dealer completes the transaction and receives payment. Finally, the pseudorandomness of the OPRF ensures that each evaluation yields a single independent output, so the party learns at most one candidate witness per execution. This implies that the party's success probability is exactly $p = 1/m$, as desired.

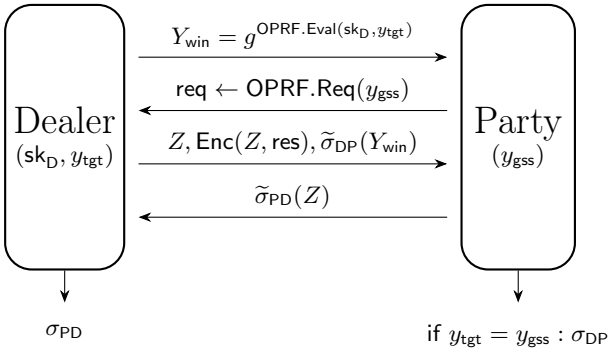


Fig. 4. Protocol overview. (1) D sends Y_win, binding y_tgt under its OPRF key. (2) P sends a blinded OPRF request for its hidden guess y_gss. (3) D returns an OPRF response encrypted under ephemeral key Z, and pre-signs P's reward w.r.t. Y_win. (4) P pre-signs D's payment w.r.t. Z; D posts it on-chain, revealing z. P extracts z, decrypts the response, and adapts $\widetilde{\sigma}_\mathsf{DP}$ to claim the reward iff $y_\mathsf{gss} = y_\mathsf{tgt}$.

3.4 Proving Well-Formedness of Y_win

The remaining technical contribution of this work is the proof of well-formedness of the statement Y_win sent by the dealer (the prover) to the party (the verifier). This proof must show that this statement admits a discrete logarithm witness that can be computed by evaluating the OPRF on inputs $(\mathsf{sk}_\mathsf{D}, y_\mathsf{tgt})$, where $y_\mathsf{tgt} \in [0, m]$. Only if the statement Y_win has this structure, the party can be sure that evaluating the OPRF on its guess y_gss will result in a probabilistic swap with the desired winning probability $p = 1/m$.

Before we talk about this proof of well-formedness, let us first recall the structure of the OPRF we are using, which is the 2HashDH OPRF [38]. With 2HashDH, the witness y_win has the structure $y_\mathsf{win} = \mathsf{H}_p\left(y_\mathsf{tgt}, \mathsf{H}_\mathbb{G}(y_\mathsf{tgt})^\mathsf{sk}\right)$. Proving well-formedness of this leads to additional challenges, as both H_p and $\mathsf{H}_\mathbb{G}$ are modeled as random oracles. We provide two solutions to this problem.

Instantiation with Bulletproofs Our first solution uses a deterministic Groth-style map-to-curve relation for $\mathsf{H}_\mathbb{G}$ and a MiMC-based hash-to-scalar relation for H_p, and shows well-formedness using R1CS in a Bulletproof [14]. We show in Section 5 that this instantiation is concretely efficient with proving time below 6 seconds, verification time below 0.35s, and proof size of 2.8KB. However, this approach has the downside of requiring the RO to be instantiated, which is unfavorable [34].

Instantiation from cut-and-choose. To avoid instantiating the random oracle in the well-formedness proof, we present a second approach based on the *cut-and-choose* technique [45]. A natural attempt is to have the dealer sample λ (statistical security parameter) target values $y_{\mathsf{tgt},1}, \ldots, y_{\mathsf{tgt},\lambda}$, compute the corresponding statements $Y_{\mathsf{win},i}$, and open a random half upon challenge. The party verifies that each opened value lies in the correct domain and that the OPRF component is well-formed with respect to the OPRF public key $\mathsf{pk} = g^\mathsf{sk}$. Cut-and-choose then guarantees that, except with negligible probability, at least one unopened instance is well-formed. As we argue next, this property is not strong enough to achieve fairness.

We observe first that the dealer must sample distinct target values $y_{\mathsf{tgt},i}$, as opening an instance reveals information about the corresponding witness. If the values are not distinct, the party could recover a valid

witness without guessing if the same target value is used in both an opened and an unopened instance. As a result, only $m - \lambda/2$ target values remain as possible winning candidates. In addition, since the party cannot decide which of the unopened statements corresponds to a valid OPRF execution, the dealer must provide a pre-signature for every unopened statement $Y_{\mathsf{win},j}$.

However, this breaks fairness: the party now holds $\lambda/2$ pre-signatures, each corresponding to a different target value, and can attempt to guess a valid witness for any of them. Since a dishonest dealer needs to make only one of the $\lambda/2$ unopened instances well-formed, the winning probability for an honest party is guaranteed to be only $p = 1/(m - \lambda/2)$. However, if the dealer behaves honestly, a possibly malicious party has a winning probability of $p = (\lambda/2)/(m - \lambda/2)$, a factor-$\lambda/2$ gap ($\lambda = 128$ in our case). This violates the core security requirement of probabilistic swaps that the winning probability $p = 1/m$ holds unconditionally, regardless of the dealer/party's behavior.

We resolve this issue by modifying the used OPRF. The difficulty is that in the 2HashDH construction, opening any instance requires revealing y_{tgt} in the clear (as it is the first part of the input to H_p), which would immediately leak the target to the party. We aim to allow opening instances *without* revealing y_{tgt}. Along this way, we first use the intermediate $\mathsf{OPRF}(\mathsf{sk}, y_{\mathsf{tgt}}) =: \mathsf{H}_p\big(g^{\mathsf{sk}}, \mathsf{H}_{\mathbb{G}}(y_{\mathsf{tgt}})^{\mathsf{sk}}\big)$, whose only modification compared to 2HashDH is the changed first input, moving from y_{tgt} to the OPRF key g^{sk}. This intermediate OPRF allows the dealer to open the outer hash H_p without revealing y_{tgt} as the first input. However, the value $\mathsf{H}_{\mathbb{G}}(y_{\mathsf{tgt}})^{\mathsf{sk}}$ is still deterministic w.r.t. y_{tgt}. Hence, we make an additional modification, which is replacing $\mathsf{H}_{\mathbb{G}}(y_{\mathsf{tgt}})^{\mathsf{sk}}$ by $\mathsf{H}_{\mathbb{G}}(y_{\mathsf{tgt}})^{\mathsf{sk}\cdot\alpha_i}$, where the values $\{g^{\alpha_i}\}_{1\le i\le\lambda}$ are additional (ephemeral) public keys for a single probabilistic swap instance. These two modifications turn out to be sufficient for our goal: For all opened instances, the prover publishes $\mathsf{H}_{\mathbb{G}}(y_{\mathsf{tgt}})^{\mathsf{sk}\cdot\alpha_i}$, and a proof of well-formedness of this value with respect to the OPRF key g^{sk}, the ephemeral key g^{α_i}, and the public set $\{\mathsf{H}_{\mathbb{G}}(y) : y \in [0, \dots, m]\}$ using a Chaum–Pedersen OR proof, without revealing y_{tgt}. For unopened instances, the dealer reveals α_j, allowing the party to locally compute and verify the candidate witness after obtaining the OPRF blind evaluation $\mathsf{H}_{\mathbb{G}}(y_{\mathsf{tgt}})^{\mathsf{sk}}$.

The resulting OPRF inherits the pseudorandomness and request privacy of the baseline 2HashDH construction, as shown in Appendix A. Since the ephemeral public keys are uniformly random, the prover can reuse the same y_{tgt} across all cut-and-choose instances. This proof strategy yields the correct winning probability of $p = 1/m$. We show in Section 7 that this cut-and-choose-based approach not only avoids instantiating the random oracle, but is also more efficient than the Bulletproof instantiation for winning probabilities $p \ge 1/2^{12}$. See Figures 17 to 20 for a formal treatment.

4 Defining Probabilistic Swaps

We formalize the security requirements of probabilistic swap protocols. The central property we aim to capture is a notion we call *probabilistic atomicity*, which can be seen as a probabilistic analog of atomic swaps [61]. We begin by recalling the functionality of deterministic atomic swaps [61], which is a simplified sub-case of probabilistic swaps (by simply setting the winning probability $p = 1$). Then, we use this baseline to explain the modifications required to obtain our probabilistic functionality.

Deterministic atomic swaps. The deterministic atomic functionality interacts with two parties over multiple rounds. In the first two rounds, each party commits to the swap by submitting their matching public inputs to the functionality. Once both parties have submitted their values and agreed to run the swap, the functionality uses their inputs to lock the respective funds under a temporary functionality-controlled address. After the funds are frozen, each party retains the ability to abort the protocol. If neither party aborts, the functionality atomically executes both transfers, paying each party. If any party aborts, the functionality releases all locked funds back to their respective owners. This mechanism ensures *deterministic* atomicity: either both transfers occur on-chain, or all funds are returned to their original owners.

Our functionality. Our functionality follows the overall structure of the deterministic case with respect to setup and funding. Both parties agree on public parameters, and the functionality locks their respective funds under a temporary, functionality-controlled address. After funding, the dealer decides whether to participate

by issuing a claim transaction. At this point, the dealer does not yet know the protocol's outcome; hence, the dealer needs to make this decision *independently* of the protocol's outcome. Once the dealer claims its payment, the functionality samples the probabilistic outcome and reveals it to the party. If the party receives a winning outcome (i.e., $b = 1$), it can claim the dealer's locked funds. Otherwise, the dealer can later reclaim its funds. If either party remains inactive, the respective other party can recover their respective funds after predefined timeouts. To retain the correct output probability for the swap, the protocol's outcome is revealed only after the dealer has committed to execution by claiming its payment. The functionality has two atomic outcomes:

– The dealer is paid. The counterparty then can claim the locked funds with probability p.
– The dealer is not paid. In this case, no exchange occurs, and both parties remain unaware of the probabilistic outcome.

In both cases, unclaimed funds can be refunded by the respective parties after an a priori fixed period. We provide a formal definition of our functionality in Figure 5.

Functionality $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$ interacts with a dealer D, and a party P.
Setup: Upon receiving $(\nu_{\mathsf{D}}, p, \mathsf{pk}_{\mathsf{D}}, \mathsf{pk}_{\mathsf{P}}, \mathbf{T}_{\mathsf{D}}, \mathbf{T}_{\mathsf{P}})$ from D, store the inputs and proceed to `Freeze`.
Fund: Upon receiving $(\nu_{\mathsf{D}}, p, \mathsf{pk}_{\mathsf{D}}, \mathsf{pk}_{\mathsf{P}}, \mathbf{T}_{\mathsf{D}}, \mathbf{T}_{\mathsf{P}})$ from P, verify that P's inputs match those provided by D. Call $\mathsf{Freeze}(\mathsf{pk}_{\mathsf{P}}, 1, \mathbf{T}_{\mathsf{P}})$ and $\mathsf{Freeze}(\mathsf{pk}_{\mathsf{D}}, \nu_{\mathsf{D}}, \mathbf{T}_{\mathsf{D}})$ and set $b = 0$. If the check and freezing succeed, send `funded` to the dealer. Otherwise, send `fail` to the dealer and the player. In both cases, store the inputs and b, and proceed to `ClaimDealer`.
Claim Dealer: Upon receiving claim from D before $\mathbf{T}_{\mathsf{P}}$ is up, set $b = 1$ with probability p, call the subroutine $\mathsf{Transfer}(1, \mathsf{pk}_{\mathsf{D}})$, and send b to P. Continue with `ClaimParty`.
Claim Party: Upon receiving claim from P before $\mathbf{T}_{\mathsf{D}}$ is up, if $b = 1$, call the subroutine $\mathsf{Transfer}(\nu_{\mathsf{D}}, \mathsf{pk}_{\mathsf{P}})$ to pay P.
Refund: Upon receiving abort from D after $\mathbf{T}_{\mathsf{D}}$ is up (P, after $\mathbf{T}_{\mathsf{P}}$), call the subroutine $\mathsf{Unfreeze}(\nu_{\mathsf{D}}, \mathsf{pk}_{\mathsf{D}})$ ($\mathsf{Unfreeze}(1, \mathsf{pk}_{\mathsf{P}})$).

Description of the subroutines. The subroutines are successful if the transaction is accepted by $\mathcal{F}_{\mathbb{B}}$.
Freeze: On input a tuple $(\mathsf{pk}, \nu, \mathbf{T})$, transfer ν coins from the address specified by pk to a script $\Lambda(\mathsf{pk}_{\mathsf{tmp}}, \mathbf{T}, \mathsf{pk})$ via $\mathtt{Post}(\mathsf{pk}, \Lambda, \nu)$, where $\mathsf{pk}_{\mathsf{tmp}}$ is a temporary address.
Transfer: On input a tuple (ν, pk) transfer ν frozen coins to the address specified by pk, via $\mathtt{Post}(\mathsf{pk}_{\mathsf{tmp}}, \mathsf{pk}, \nu)$.
Unfreeze: On input (ν, pk), move ν coins from Λ to pk, making them spendable only by pk via $\mathtt{Post}(\mathsf{pk}, \mathsf{pk}, \nu)$.

Fig. 5. The ideal functionality $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$.

Other key differences with the functionality of [61]. While our functionality follows the high-level structure of [61], we introduce several modifications that facilitate practical realizations.

Refunds. The functionality of [61] assumes that funds can be atomically unfrozen. In most protocols, however, refunding requires active participation of the party controlling the locked coins (e.g., by issuing a transaction). In particular, an adversary in the security experiment could act irrationally by refusing to post a refund transaction, thereby breaking the simulation. To capture this more faithfully, our functionality models refunds explicitly: each party must trigger its own refund separately.

Timing-based locking. The functionality of [61] does not model time, and funds remain locked until they are explicitly released. In contrast, our functionality is parameterized by timeouts $\mathbf{T}_{\mathsf{D}}$ and $\mathbf{T}_{\mathsf{P}}$, after which the dealer and the party, respectively, can reclaim their funds. The main motivation for this change is that practical atomic swap protocols are typically realized using timelock scripts rather than verifiable timed signatures (VTS) as in [61]. Explicitly modeling time avoids mismatches between ideal and real protocol executions, since in the real execution, a locking script would be the recipient of the freezing transaction, whereas in the simulated world, a temporary address would be. We emphasize that this is primarily a

modeling choice: our functionality could equivalently be instantiated using a timeless approach based on VTS.

Public inputs only. In contrast to [61], our functionality does not require parties to provide secret inputs, but only public parameters of the exchange. This significantly simplifies simulation, as the simulator no longer needs to extract secret keys from the adversary, which would otherwise require additional proofs of knowledge. At first glance, this may appear at odds with the fact that blockchain transactions require signatures. However, both our blockchain functionality and that of [61] model the blockchain as an ordered list of transactions, abstracting away transaction validation. Since the ideal blockchains accept all incoming transactions, the proofs simulator that sits between the adversary and the ideal blockchain must ensure that only authorized transactions are sent to the blockchain. We do this by only appending signed transactions to the blockchain.

Handling of signatures. Both our protocol and that of [61] rely on signatures in concrete realizations. However, signatures are not explicitly modeled by the blockchain functionality. To resolve this issue, we incorporate them at the simulation level. Concretely, the simulator only appends transactions to the blockchain functionality if the corresponding party provides a valid signature. Moreover, when the adversary observes the blockchain, the simulator augments the output with the corresponding signatures before forwarding it to the adversary. Not handling signatures like this would be an issue in the proof of [61] as well, so we conclude that the authors assume this is done implicitly.

5 Our Protocol

We present our probabilistic swap protocol between a dealer D and a party P. As building blocks, we use adaptor signatures and OPRFs, which we recalled in Section 2. Additionally, we rely on secure two-party computations for the adaptor signature key pairs and pre-signing. For ease of exposition, we focus on Schnorr pre-signatures with DLog keypairs. However, the respective sub-protocols in Figures 7 and 8 could simply be replaced by secure two-party protocols for any adaptor signature scheme.

2-party DLog key generation. We use a standard two-party protocol for generating a shared discrete logarithm keypair, following [25]. The protocol ensures that both parties obtain additive secret shares of a jointly generated public key, while proving knowledge of their respective shares. We depict the corresponding ideal functionality in Figure 6. It allows the adversary to fix its secret share while the honest party's share is

Functionality $\mathcal{F}_{\mathsf{DLog}}$ interacts with an honest party P and an adversary $\mathcal{A}$.

1. Receive $\mathsf{sk}_{\mathcal{A}}$ from the adversary.
2. Choose $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$ and set $\mathsf{pk} = g^{\mathsf{sk}}$.
3. Send pk to $\mathcal{A}$. If $\mathcal{A}$ sends $\mathtt{ok}$, send $(\mathsf{pk}, \mathsf{sk} - \mathsf{sk}_{\mathcal{A}})$ to P.

Fig. 6. Two-party DLog keypair generation functionality.

sampled uniformly at random. In particular, this enables us to learn the adversarial share of the secret key while determining the joint public key, which will be crucial in the security proof. We formalize the standard protocol realizing this two-party key generation functionality in Figure 7.

2-party Schnorr pre-signing. We use a two-party variant of Schnorr pre-signing as introduced in [4], which allows two parties holding additive secret shares to jointly produce a valid Schnorr pre-signature. On public input, a public key pk, a transaction tx, and a statement Y, and secret inputs being the key shares of two parties corresponding to pk, the protocol outputs a valid pre-signature. We formally recall the protocol in Figure 8.

Global input: $(\mathbb{G}, g, p)$ and a hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p$. Parties have no individual inputs.

1. Party P_0 samples a secret key $\mathsf{sk}_0 \leftarrow\!\!\$\ \mathbb{Z}_p$, the corresponding public key $\mathsf{pk}_0 = g^{\mathsf{sk}_0}$, commits to pk_0 via $\mathsf{C}_0 = \mathsf{H}(\mathsf{pk}_0)$, and sends C_0 to party P_1.
2. Party P_1 samples a secret key $\mathsf{sk}_1 \leftarrow\!\!\$\ \mathbb{Z}_p$, the corresponding public key $\mathsf{pk}_1 = g^{\mathsf{sk}_1}$ and proves knowledge of sk_1 via $\pi_1 \leftarrow \Pi.\mathcal{P}(\mathsf{pk}_1, \mathsf{sk}_1)$. It sends (pk_1, π_1) to P_0.
3. Party P_0 receives (pk_1, π_1). If the proof verifies, i.e., $\Pi.\mathcal{V}(\mathsf{pk}_1, \pi_1) = 1$, party P_0 also proves knowledge of sk_0 via $\pi_0 \leftarrow \Pi.\mathcal{P}(\mathsf{pk}_0, \mathsf{sk}_0)$ and sends (pk_0, π_0) to P_1. Otherwise, it aborts.
4. Party P_1 asserts, that pk_0 is a valid opening for C_0 by checking if $\mathsf{H}(\mathsf{pk}_0) = \mathsf{C}_0$ and verifies the proof by checking if $\Pi.\mathcal{V}(\mathsf{pk}_0, \pi_0) = 1$. It aborts otherwise.

Party P_0 sets its output to $(\mathsf{pk}_0 \cdot \mathsf{pk}_1, \mathsf{sk}_0)$. Party P_1 sets its output to $(\mathsf{pk}_0 \cdot \mathsf{pk}_1, \mathsf{sk}_1)$.

Fig. 7. Two-Party protocol Γ_{DL} for generating a DLog keypair. C.f. Figure 16 for the corresponding proof system.

Global input: $(\mathsf{pk}, \mathsf{tx}, Y)$. Party P_0's input: sk_0. P_1's input: sk_1.

1. Both parties run the two-party protocol from Figure 7 to obtain a joint value R, while P_0 receives r_0 and P_1 receives r_1, such that $R = g^{r_0+r_1}$.
2. Party P_0 computes $s_0 = \mathsf{sk}_0 \cdot c + r_0$ and sends s_0 to P_1, where $c = \mathsf{H}(\mathsf{pk}, R \cdot Y, \mathsf{tx})$.
3. Party P_1 receives s_0 from P_0 and computes $s_1 = \mathsf{sk}_1 \cdot c + r_1$, where $c = \mathsf{H}(\mathsf{pk}, R \cdot Y, \mathsf{tx})$, and sends s_1 to P_0.

Both parties set their output to $(R \cdot Y, s_0 + s_1)$. P_1 receives the output first.

Fig. 8. Two-Party Schnorr pre-signing protocol Γ_{pSign}.

Protocol description. Using the above building blocks, we can describe our probabilistic atomic swap protocol. The protocol reads as public input a buyout amount ν_{D}, a winning probability $p = 1/m$, the dealer's public key pk_{D}, the counterparty's public key pk_{P}, and two timeout parameters $\mathbf{T}_{\mathsf{D}}$ and $\mathbf{T}_{\mathsf{P}}$. The protocol's output is a reallocation of the dealer's and player's funds, as depicted in Figure 1. We require that $\mathbf{T}_{\mathsf{D}} > \mathbf{T}_{\mathsf{P}}$, ensuring that once the dealer has issued its claim, the party has sufficient time $\mathbf{T}_{\mathsf{D}} - \mathbf{T}_{\mathsf{P}}$ to submit its own claim. Since our construction relies on timelock scripts on the blockchain rather than verifiable timed signatures (VTS) as in [61], the timeout parameters need not be chosen overly conservatively, which provides the dealer with faster liquidity if the party loses. We provide our protocol in Figure 9, which realizes probabilistic swaps for probabilities of the form $1/m$. However, this baseline protocol can be extended to support arbitrary rational probabilities in $\mathbb{Q}$, as discussed in Section 6. We omit a high-level overview of our protocol here as we provided one in Section 3.

Theorem 1. *Let* AS *be an extractable and adaptable adaptor signature scheme w.r.t. a strongly unforgeable signature scheme* Σ*, and a hard relation* $\mathcal{R}$*. Let* Γ_{KG} *and* Γ_{pSign} *be secure 2PC pre-signing and key generation protocols for* AS*. Let* Π_{Enc} *be a CPA-secure encryption scheme with key pairs matching the relation of* AS*, and* Π *a knowledge sound zero-knowledge proof system. Let* OPRF *be a pseudorandom and request-private two-hash OPRF mapping in the witness space of* $\mathcal{R}$*. Then, for all* $\nu_{\mathsf{D}} \in \mathbb{Q}^+, m \in \mathbb{N}$*, and* $p = 1/m$*, the probabilistic atomic swap protocol described in Figure 9 with access to* $\mathcal{F}_{\mathbb{B}}$ *securely realizes* $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$*, i.e., for every PPT adversary* $\mathcal{A}$ *there exists a PPT simulator* Sim *such that*

$$\mathtt{IDEAL}_{\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}},\mathsf{Sim}} \approx_c \mathtt{REAL}_{\Pi,\mathcal{A}}.$$

We refer to Section 3.3 for an intuitive proof description. The central component of our protocol, which we referred to as *OPRF evaluation as a service* in Section 3, is implemented in the claim-dealer phase. There, the dealer acts as an OPRF server: it blindly evaluates the OPRF on the party's guess, verifiably encrypts

Global input: $(\nu_{\mathsf{D}}, p, m, \mathsf{pk}_{\mathsf{D}}, \mathsf{pk}_{\mathsf{P}}, \mathbf{T}_{\mathsf{D}}, \mathbf{T}_{\mathsf{P}})$, dealer's input: sk_{D}, player's input: sk_{P}. It holds that $p = 1/m$.
Setup: The dealer verifiably encrypts the winning witness under the (small domain) target witness.

1. Sample the target witness $y_{\mathsf{tgt}} \leftarrow\!\!\$\ [0, m]$, and compute $y_{\mathsf{win}} \leftarrow \mathsf{OPRF.Eval}(\mathsf{sk}_{\mathsf{D}}, y_{\mathsf{tgt}})$, $Y_{\mathsf{win}} = g^{y_{\mathsf{win}}}$.
2. Prove well-formedness of Y_{win} via $\pi \leftarrow \Pi.\mathcal{P}(\mathcal{R}_{\mathsf{win}}, (\mathsf{pk}_{\mathsf{D}}, Y_{\mathsf{win}}, m), (\mathsf{sk}_{\mathsf{D}}, y_{\mathsf{tgt}}))$ and send (π, Y_{win}) to the party. We define the corresponding relation language $\mathcal{R}_{\mathsf{win}}$ in Appendix B.

Funding Phase: The dealer and the party lock the coins they want to use in the protocol.

1. They run the key generation protocol of Figure 7. Joint output: $\mathsf{pk}_{\mathsf{tmp}}$. Private output for dealer: $\mathsf{sk}_{\mathsf{tmp,D}}$, for the party: $\mathsf{sk}_{\mathsf{tmp,P}}$.
2. Both party and dealer create locking transactions $\mathsf{tx}_{\mathsf{D}} = (\mathsf{pk}_{\mathsf{D}}, \nu_{\mathsf{D}}, \Lambda(\mathsf{pk}_{\mathsf{tmp}}, \mathbf{T}_{\mathsf{D}}, \mathsf{pk}_{\mathsf{D}}))$ and $\mathsf{tx}_{\mathsf{P}} = (\mathsf{pk}_{\mathsf{P}}, 1, \Lambda(\mathsf{pk}_{\mathsf{tmp}}, \mathbf{T}_{\mathsf{P}}, \mathsf{pk}_{\mathsf{P}}))$ that spend ν_{D} (one) coins from the address corresponding to pk_{D} (pk_{P}) and whose outputs are the locking scripts Λ (c.f., Figure 2).
3. The dealer spends these locking transactions by computing $\sigma_{\mathsf{D}} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{D}}, \mathsf{tx}_{\mathsf{D}})$ and sending $(\mathsf{tx}_{\mathsf{D}}, \sigma_{\mathsf{D}})$ to the blockchain $\mathbb{B}$. The party also spends the locking transactions with $(\mathsf{tx}_{\mathsf{P}}, \sigma_{\mathsf{P}})$, where $\sigma_{\mathsf{P}} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{P}}, \mathsf{tx}_{\mathsf{P}})$.

When funding is complete, the dealer's funds can be spent solely by $\mathsf{pk}_{\mathsf{tmp}}$ for $\mathbf{T}_{\mathsf{D}}$ timeout period. Afterwards, both $\mathsf{pk}_{\mathsf{tmp}}$ and pk_{D} can spend it. The same holds for the party's funds with $\mathbf{T}_{\mathsf{P}}$ and pk_{P}.
Claim Dealer: After the swap is successfully funded, the dealer and party exchange information until the dealer is paid.

1. The party samples $y_{\mathsf{gss}} \leftarrow\!\!\$\ [0, m]$, computes an OPRF request $(\mathsf{st}, \mathsf{req}) \leftarrow \mathsf{OPRF.Req}(y_{\mathsf{gss}})$, and sends req to the dealer.
2. The dealer evaluates the OPRF request $\mathsf{res} \leftarrow \mathsf{OPRF.BlindEv}(\mathsf{sk}, \mathsf{req})$, samples an encryption key $(Z, z) \leftarrow \mathsf{KGen}(\lambda)$, and verifiably encrypts res under Z via $ct_{\mathsf{D}} \leftarrow \Pi_{\mathsf{Enc}}.\mathsf{Enc}(Z, \mathsf{res})$, $\pi_{\mathsf{D}} \leftarrow \Pi.\mathcal{P}(\mathcal{R}_{\mathsf{Enc}}, (\mathsf{pk}_{\mathsf{D}}, Z, \mathsf{req}, ct_{\mathsf{D}})(\mathsf{sk}_{\mathsf{D}}, z))$. We define the corresponding relation language $\mathcal{R}_{\mathsf{Enc}}$ in Appendix B. Finally, the dealer sends $(Z, ct_{\mathsf{D}}, \pi_{\mathsf{D}})$ to the party.
3. Dealer and party jointly pre-sign the transactions $\mathsf{tx}_{\mathsf{PD}} = (\mathsf{pk}_{\mathsf{tmp}}, 1, \mathsf{pk}_{\mathsf{D}})$ which spends one coin from $\mathsf{pk}_{\mathsf{tmp}}$ to pk_{D} and $\mathsf{tx}_{\mathsf{DP}} = (\mathsf{pk}_{\mathsf{tmp}}, \nu_{\mathsf{D}}, \mathsf{pk}_{\mathsf{P}})$ that spends ν_{D} coins from $\mathsf{pk}_{\mathsf{tmp}}$ to pk_{P} by running the pre-signing protocol of Figure 8 twice.
   - Joint input: $(\mathsf{tx}_{\mathsf{DP}}, Y_{\mathsf{win}})$. Private input: D inputs $\mathsf{sk}_{\mathsf{tmp,D}}$, party inputs $\mathsf{sk}_{\mathsf{tmp,P}}$. The protocol outputs $\widetilde{\sigma}_{\mathsf{DP}}$ first to D then to P.
   - Joint input: $(\mathsf{tx}_{\mathsf{PD}}, Z)$. Private input: D inputs $\mathsf{sk}_{\mathsf{tmp,D}}$, party inputs $\mathsf{sk}_{\mathsf{tmp,P}}$. The protocol outputs $\widetilde{\sigma}_{\mathsf{PD}}$ first to P then to D.
4. The dealer adapts $\widetilde{\sigma}_{\mathsf{PD}}$ using z via $\sigma_{\mathsf{PD}} \leftarrow \mathsf{AS.Adapt}(\mathsf{pk}_{\mathsf{P}}, \widetilde{\sigma}_{\mathsf{PD}}, z)$ and posts $(\mathsf{tx}_{\mathsf{PD}}, \sigma_{\mathsf{PD}})$ to the blockchain.

Claim Party: The party claims the ν_{D} coins if its guess was correct.

1. The party extracts z from σ_{PD} obtained from $\mathbb{B}$ via $z \leftarrow \mathsf{AS.Extract}(Z, \widetilde{\sigma}_{\mathsf{PD}}, \sigma_{\mathsf{PD}})$.
2. It decrypts the OPRF response via $\mathsf{res} \leftarrow \Pi_{\mathsf{Enc}}.\mathsf{Dec}(z, ct)$ and computes the guessed witness via $y'_{\mathsf{win}} \leftarrow \mathsf{OPRF.Finalize}(\mathsf{res}, \mathsf{st})$.
3. If the guess was correct, the party obtains $\sigma_{\mathsf{DP}} \leftarrow \mathsf{AS.Adapt}(\mathsf{pk}_{\mathsf{tmp}}, \widetilde{\sigma}_{\mathsf{DP}}, y'_{\mathsf{win}})$, and posts $(\mathsf{tx}_{\mathsf{DP}}, \sigma_{\mathsf{DP}})$ to the blockchain $\mathbb{B}$.

Timeout Phase: Parties reclaim their frozen funds if the protocol fails or the other party did not claim their coins.

1. If the dealer fails to post $(\mathsf{tx}_{\mathsf{PD}}, \sigma_{\mathsf{PD}})$ to $\mathbb{B}$ within time $\mathbf{T}_{\mathsf{P}}$, the party creates a refund transaction $\mathsf{tx}_{\mathsf{rfndP}} = (\mathsf{pk}_{\mathsf{P}}, 1, \mathsf{pk}_{\mathsf{P}})$ that spends the party's unspent outputs from $\mathsf{tx}_{\mathsf{PD}}$, signs it via $\sigma_{\mathsf{rfndP}} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{P}}, \mathsf{tx}_{\mathsf{rfndP}})$ and sends $(\mathsf{tx}_{\mathsf{rfndP}}, \sigma_{\mathsf{rfndP}})$ to the blockchain $\mathbb{B}$.
2. Similarly, if the party fails to post $(\mathsf{tx}_{\mathsf{DP}}, \sigma_{\mathsf{DP}})$ to $\mathbb{B}$ within time $\mathbf{T}_{\mathsf{D}}$, the dealer creates a refund transaction $\mathsf{tx}_{\mathsf{rfndD}} = (\mathsf{pk}_{\mathsf{D}}, \nu_{\mathsf{D}}, \mathsf{pk}_{\mathsf{D}})$, signs it via $\sigma_{\mathsf{rfndD}} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{D}}, \mathsf{tx}_{\mathsf{rfndD}})$ and sends $(\mathsf{tx}_{\mathsf{rfndD}}, \sigma_{\mathsf{rfndD}})$ to the blockchain $\mathbb{B}$.

Fig. 9. Our probabilistic atomic swap protocol from adaptor signatures and oblivious pseudorandom functions for exchange probabilities in the powers of two.

the response, and uses an adaptor pre-signature to tie the release of the decryption key atomically to its own payment. This treatment of OPRF evaluation as an atomically purchasable service is what upgrades the probabilistic guessing mechanism into a probabilistic atomic swap.

Proof (Proof of Theorem 1). To prove Theorem 1, we show that for every PPT adversary $\mathcal{A}$ corrupting either the dealer or the party, there exists an efficient simulator Sim that simulates the real-world protocol execution with $\mathcal{A}$ while interacting with the ideal functionality $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$. We consider static corruptions, where the environment $\mathcal{E}$ specifies the corrupted party at the beginning of the session. The adversary controls the corrupted party, while the simulator is given the corrupted party's internal state, in particular its secret key. The simulator faithfully impersonates the honest party. As we carry out the proof in the $\mathcal{F}_{\mathsf{smt}}$-hybrid world, communication reveals only that it occurred, without leaking any information about message contents. Therefore, we omit modelling communication in the simulator. We proceed in two steps, constructing one simulator for the case of a corrupted dealer and one for the case of a corrupted party.

We begin with the case where the adversary corrupts the party and construct a simulator $\mathsf{Sim}_{\mathsf{D}}$. We analyze this case via a sequence of hybrid executions, starting from the real-world execution and gradually modifying it until we obtain the simulator's execution. We then argue that any two consecutive hybrids are indistinguishable. For simplicity, we describe a single session, noting that the argument extends to multiple sessions in a straightforward manner.

Exp. 0. The first experiment is the honest execution of our protocol depicted in Figure 9.

Exp. 1. In Exp. 1, we replace honestly computed zero-knowledge proofs by their simulated counterparts using the simulator $\mathsf{Sim}_{\mathsf{NIZK}}$. By the zero-knowledge property of Π, there exists such a simulator. $\mathsf{Sim}_{\mathsf{NIZK}}$ reads a statement $\mathbb{x}$ and outputs a simulated proof π.

Exp. 2. In Exp. 2, we replace the key generation protocol of Figure 7 using the 2PC simulator $\mathsf{Sim}_{\mathsf{DKG}}$. Such a simulator exists for this protocol, as our theorem assumes it is a secure 2PC protocol. $\mathsf{Sim}_{\mathsf{DKG}}$ reads a key $\mathsf{pk}_{\mathsf{tmp}} \in \mathbb{G}$ and outputs $\mathsf{sk}_{\mathsf{tmp,P}}$ to $\mathsf{Sim}_{\mathsf{D}}$.

Exp. 3. In Exp. 3, we replace the joint adaptor signing protocols of Figure 8 using the 2PC simulator $\mathsf{Sim}_{\mathsf{pSign}}$. Such a simulator exists for this protocol, as our theorem assumes it is a secure 2PC protocol. $\mathsf{Sim}_{\mathsf{pSign}}$ reads a public key $\mathsf{pk}_{\mathsf{tmp}} \in \mathbb{G}$, a secret key $\mathsf{sk}_{\mathsf{tmp,P}}$, and a pre-signature $\widetilde{\sigma}$ and has no output to $\mathsf{Sim}_{\mathsf{D}}$. $\mathsf{Sim}_{\mathsf{pSign}}$ ensures, that the pre-signature computed in the 2PC equals $\widetilde{\sigma}$.

Exp. 4. In Exp. 4, we replace all OPRF-related operations from the dealer using the simulator $\mathsf{Sim}_{\mathsf{OPRF}}$. Such a simulator exists, as our theorem assumes that the OPRF is pseudorandom. $\mathsf{Sim}_{\mathsf{OPRF}}$ reads public parameters, outputs a public key pk, and provides access to $\mathsf{Eval}, \mathsf{BlindEval}, \mathsf{Prim}$ oracles.

Exp. 5. In Exp. 5, the simulator aborts with message abort_0 if the adversary attempts to post a tuple $(\mathsf{tx}^*, \sigma^*)$ to $\mathbb{B}$, in which σ^* is valid w.r.t. pk_{D} on tx^*, and $\mathsf{tx}^* \notin \{\mathsf{tx}_{\mathsf{D}}, \mathsf{tx}_{\mathsf{rfndD}}\}$.

Exp. 6. In Exp. 6, the simulator aborts with message abort_1 if the adversary attempts to post a tuple $(\mathsf{tx}^*, \sigma^*)$ to $\mathbb{B}$, in which σ^* is valid w.r.t. $\mathsf{pk}_{\mathsf{tmp}}$ on tx^*, and σ^* does not allow extracting a valid witness for Y_{win} from $\widetilde{\sigma}_{\mathsf{DP}}$, i.e., $(Y_{\mathsf{win}}, \mathsf{AS.Extract}(\mathsf{pk}_{\mathsf{tmp}}, \widetilde{\sigma}_{\mathsf{DP}}, \sigma^*, Y_{\mathsf{win}})) \notin \mathcal{R}$. If our protocol did not yet define $\widetilde{\sigma}_{\mathsf{DP}}$, we set it to $\perp$.

Exp. 7. In Exp. 7, the simulator aborts with message abort_2 if the adversary attempts to post a tuple $(\mathsf{tx}^*, \sigma^*)$ to $\mathbb{B}$, in which σ^* is valid w.r.t. $\mathsf{pk}_{\mathsf{tmp}}$ on tx^*, *before* the simulator forwards σ_{PD} to $\mathcal{A}$.

Exp. 8. In Exp. 8, at the beginning of the experiment, the simulator uses the two-hash structure of the OPRF, and whenever the adversary queries the Prim oracle for the first hash function H_1 on some input $y_{\hat{\mathsf{gss}}} \in [2^\ell]$, the simulator computes $y = F(\mathsf{sk}, H_1(y_{\hat{\mathsf{gss}}})$, and stores $(y_{\hat{\mathsf{gss}}}, y)$. We denote these tuples as *valid OPRF preimages*. The simulator aborts with message abort_3 if an input to the Prim oracle is a valid OPRF preimage, but the simulator did not forward σ_{PD} to $\mathcal{A}$, or if this output would be the second valid OPRF preimage to the Prim oracle to the adversary.

Exp. 9. In Exp. 9, the simulator changes the way it simulates Prim to $\mathcal{A}$. When the adversary queries Prim, on input z, $\mathsf{Sim}_{\mathsf{D}}$ checks if this is a valid OPRF preimage (defined in Exp. 8). If this is the case *after* the functionality outputs $b = 1$, the simulator changes the output of Prim to $\mathsf{Prim}(z) = y_{\mathsf{win}}$.

Simulator $\mathsf{Sim}_{\mathsf{D}}$. The execution of the simulator is defined as the execution in Exp. 9. The simulator $\mathsf{Sim}_{\mathsf{D}}$ reads as input $(\nu_{\mathsf{D}}, p, \ell, \mathbf{T}_{\mathsf{D}}, \mathbf{T}_{\mathsf{P}})$ and starts by internally running $\mathsf{Sim}_{\mathsf{OPRF}}$ to receive the OPRF-part $\mathsf{pk}_{\mathsf{D,OPRF}}$

of pk_D. Second, it runs $(\mathsf{pk}_{\mathsf{D},\Sigma}, \mathsf{sk}_{\mathsf{D},\Sigma}) \leftarrow \mathsf{KGen}(\lambda)$ to obtain the signature part of pk_D. Then, Sim_D internally executes $\mathcal{A}$ on input $(\nu_\mathsf{D}, p, \ell, \mathbf{T}_\mathsf{D}, \mathbf{T}_\mathsf{P}, \mathsf{pk}_\mathsf{D})$. Eventually, $\mathcal{A}$ outputs pk_P.
Setup: Sim_D samples $(Y_\mathsf{win}, y_\mathsf{win}) \leftarrow\!\!\$\ \mathsf{genR}(\lambda)$, and simulates π using $\mathsf{Sim}_\mathsf{NIZK}$ on input $\mathbb{x} = (\mathsf{pk}_\mathsf{D}, Y_\mathsf{win}, \ell)$. The simulator sends (Y_win, π) to $\mathcal{A}$, and $(\nu_\mathsf{D}, p, \mathsf{pk}_\mathsf{D}, \mathsf{pk}_\mathsf{P}, \mathbf{T}_\mathsf{D}, \mathbf{T}_\mathsf{P})$ to $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$.
Funding: $\mathcal{A}$ eventually attempts to run the DKG protocol. To run this protocol, Sim_D first samples a keypair $(\mathsf{pk}_\mathsf{tmp}, \mathsf{sk}_\mathsf{tmp}) \leftarrow \mathsf{KeyGen}(\lambda)$ and interacts with $\mathcal{A}$ using $\mathsf{Sim}_\mathsf{DKG}$ on input pk_tmp. Eventually, $\mathcal{A}$ receives back $\mathsf{sk}_{\mathsf{tmp},\mathsf{P}}$. After running this DKG, the adversary eventually attempts to post $(\mathsf{tx}_\mathsf{P}, \sigma_\mathsf{P})$ on-chain. The simulator computes a signature σ_D and provides the pair $(\mathsf{tx}_\mathsf{D}, \sigma_\mathsf{D})$ to $\mathcal{A}$. In addition, the simulator invokes the `Freeze` interface of $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$.
Claim Dealer: Eventually, $\mathcal{A}$ sends a request req to the simulator. Sim_D then computes the response $\mathsf{res} \leftarrow \mathsf{Sim}_{\mathsf{OPRF}}.\mathsf{BlindEv}(\mathsf{req})$, samples $(Z, z) \leftarrow \mathsf{genR}(\lambda)$, and $ct_\mathsf{D} \leftarrow \Pi_{\mathsf{Enc}}.\mathsf{Enc}(Z, \mathsf{res})$. Finally, Sim_D simulates π_D using $\mathsf{Sim}_\mathsf{NIZK}$ on input $\mathbb{x} = (\mathsf{pk}_\mathsf{D}, Z, \mathsf{req}, ct_\mathsf{D})$ and forwards $(Z, ct_\mathsf{D}, \pi_\mathsf{D})$ to $\mathcal{A}$. Then, to run both pre-signing protocols, Sim_D locally pre-signs each $\mathsf{tx}_{\mathsf{PD}}, \mathsf{tx}_{\mathsf{DP}}$ using sk_tmp, resulting in $\widetilde{\sigma}_{\mathsf{PD}}$, $\widetilde{\sigma}_{\mathsf{DP}}$, and uses $\mathsf{Sim}_{\mathsf{pSign}}$ on input $(\mathsf{sk}_{\mathsf{tmp},\mathsf{P}}, \widetilde{\sigma}_{\mathsf{PD}})$ and $(\mathsf{sk}_{\mathsf{tmp},\mathsf{P}}, \widetilde{\sigma}_{\mathsf{DP}})$, to enforce these pre-signatures as outputs of the pre-signing protocols. Finally, Sim_D adapts $\widetilde{\sigma}_{\mathsf{PD}}$ using z, and outputs σ_{PD} to $\mathcal{A}$. In addition, Sim_D calls the `ClaimDealer` interface of $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$, which responds with a bit b.
Claim Party: Eventually, $\mathcal{A}$ queries the Prim interface on a valid OPRF preimage. If this is the case, and $b = 1$, Sim_D returns y_win to $\mathcal{A}$. If $\mathcal{A}$ attempts to send $(\mathsf{tx}_{\mathsf{DP}}, \sigma_{\mathsf{DP}})$ to the blockchain, Sim_D calls the `ClaimParty` interface of $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$.
Timeout: If funding was successful, but $\mathcal{A}$ did not participate in the protocol until $\mathbf{T}_\mathsf{D}$ is up, Sim_D signs $\mathsf{tx}_{\mathsf{rfndD}}$, sends abort to $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$, and forwards $(\mathsf{tx}_{\mathsf{rfndD}}, \sigma_{\mathsf{rfndD}})$ to $\mathcal{A}$.

On top of this simulation, Sim_D aborts if the adversary forges signatures under pk_D (c.f., Exp. 5) or pk_tmp (c.f., Experiments 6 and 7), or obtains unfaithful OPRF evaluations (c.f., Exp. 8).

We next show that the differences between two neighboring experiments are computationally indistinguishable. Since we have only polynomially many experiments, this establishes the first part of our theorem, since by this negligible transition, we have

$$\mathtt{IDEAL}_{\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}},\mathsf{Sim}_\mathsf{D}} \approx \mathtt{REAL}_{\Pi^{\mathbb{B}}_{\mathsf{ProSwap}},\mathcal{A}_\mathsf{P}}.$$

Exp. 0 ≈ Exp. 1. These experiments differ only in the way how zero-knowledge proofs are computed by Sim_D. By the zero-knowledge property of Π, indistinguishability follows canonically.
Exp. 1 ≈ Exp. 2. These experiments differ only in how the two-party DKG protocol is executed. In Exp. 1, Sim_D runs the honest DKG protocol of Figure 7 with $\mathcal{A}$. In Exp. 2, it is replaced by the 2PC simulator $\mathsf{Sim}_\mathsf{DKG}$. Since Theorem 1 assumes Γ_KG is a secure two-party protocol, such a simulator exists, and the adversary's view is computationally indistinguishable across the two experiments.
Exp. 2 ≈ Exp. 3. These experiments differ only in how the two joint pre-signing protocols of Figure 8 are executed. In Exp. 2, Sim_D runs the honest pre-signing protocol. In Exp. 3, it is replaced by the 2PC simulator $\mathsf{Sim}_\mathsf{pSign}$. Since Theorem 1 assumes Γ_pSign is a secure two-party protocol, the adversary's view is computationally indistinguishable.
Exp. 3 ≈ Exp. 4. These experiments differ in the way how OPRF queries are handled by Sim_D. If an efficient adversary could distinguish this change, it would by definition break the pseudorandomness of OPRF.
Exp. 4 ≈ Exp. 5. Indistinguishability between these experiments follows canonically from the unforgeability of Σ.
Exp. 5 ≈ Exp. 6. We show the computational indistinguishability between Exp. 5 and Exp. 6 by building a reduction $\mathcal{B}$ that uses an efficient distinguisher of these experiments and breaks the extractability of AS. As input, $\mathcal{B}$ receives a public key pk, and it sets $\mathsf{pk}_\mathsf{tmp} = \mathsf{pk}$. In addition, $\mathcal{B}$ has access to a pre-signing oracle and a signing oracle that output (pre-)signatures valid on pk_tmp on messages (and statements) of $\mathcal{B}$'s choice. Whenever Sim_D needs to pre-sign, it forwards the respective message-statement pair to the pre-sign oracle. During simulation, Sim_D never needs to sign messages using sk_tmp. With these modifications, $\mathcal{B}$ simulates Exp. 4 to $\mathcal{A}$. By assumption, there exists an efficient distinguisher between the experiments. As the experiments only differ, if the adversary attempts to post a tuple $(\mathsf{tx}^*, \sigma^*)$ to $\mathbb{B}$, in which σ^* is valid w.r.t. pk_tmp on tx^*, and σ^* does not allow extracting a valid witness for Y_win from $\widetilde{\sigma}_{\mathsf{DP}}$, i.e., $(Y_\mathsf{win}, \mathsf{AS.Extract}(\mathsf{pk}_\mathsf{tmp}, \widetilde{\sigma}_{\mathsf{DP}}, \sigma^*, Y_\mathsf{win})) \notin \mathcal{R}$,

this must be the case. Otherwise, the experiments would be perfectly indistinguishable. If the adversary outputs such a forgery, $\mathcal{B}$ outputs it as well. By definition, this breaks the extractability of AS. In addition, the simulation of $\mathcal{B}$ is perfect, as we replace pre-signing with perfect oracles and efficient, as Exp. 4 is efficient.

Exp. 6 ≈ Exp. 7. We demonstrate the computational closeness between these two experiments using a subexperiment. This subexperiment is idential to Exp. 6, but instead of encrypting the response res in ct_{D}, $\mathsf{Sim}_{\mathsf{D}}$ encrypts the generator g. We show, that before $\mathsf{Sim}_{\mathsf{D}}$ forwards σ_{DP} to $\mathcal{A}$, Exp. 6 and this subexperiment are computationally indistinguishable. We do this by building a reduction $\mathcal{B}$ from the distinguishability between these two experiments before $\mathsf{Sim}_{\mathsf{D}}$ sends σ_{DP}, and in which $\mathcal{A}$ posts such a forgery to $\mathcal{A}$ to the CPA security of Π_{Enc}. In other words, by contradiction, we assume the existence of an efficient distinguisher $\mathcal{D}$. As input $\mathcal{B}$ receives a public key pk, and sets $Z = \mathsf{pk}$. Since we only consider the experiment before $\mathsf{Sim}_{\mathsf{D}}$ sends the signature, it is sufficient for $\mathcal{B}$ to not know z. Then, $\mathcal{B}$ simulates Exp. 6. Instead of encrypting the response res, it forwards the pair (res, g) to its CPA challenger, and receives back a ciphertext ct. $\mathcal{B}$ continues simulating Exp. 6 with $ct_{\mathsf{D}} = ct$ and outputs the transcript of the experiment to $\mathcal{D}$. Eventually, $\mathcal{D}$ outputs a bit and $\mathcal{B}$ outputs the same bit. If the CPA challenger encrypts res, $\mathcal{B}$ simulated Exp. 6 perfectly to $\mathcal{D}$. In the other case, it simulated our subexperiment. Therefore, $\mathcal{D}$'s advantage in distinguishing the experiments carries over to $\mathcal{B}$'s ability to break CPA.

Next, we show that this subexperiment and Exp. 7 are computationally indistinguishable by the hardness of the relation $\mathcal{R}$. This holds, since in the situation of Exp. 6, the adversary can only compute signature forgeries that allow it to extract a valid witness for Y_{win}. In the situation of our subexperiment, the adversary learns no OPRF evaluation by running the protocol. Therefore, we can compute a reduction $\mathcal{B}$ that receives Y_{win}, simulates our subexperiment to $\mathcal{A}$ (note that knowledge of y_{win} is not required to do so, since the abort happens), and if the adversary triggers the additional abort in Exp. 7, $\mathcal{B}$ learns a valid witness for Y_{win} and breaks the hardness of $\mathcal{R}$. This reduction is perfect, since a uniformly random Y_{win} is equally distributed to the way our protocol computes Y_{win}, as in the situation of Exp. 4, the OPRF is a random function. Combining these two steps, the closeness between Experiments 6 and 7 holds.

Exp. 7 ≈ Exp. 8. Both cases in which Exp. 8 aborts can only happen if the adversary obtains a valid OPRF preimage (and hence a correct OPRF evaluation) without querying the $\mathsf{BlindEv}$ oracle. By the pseudorandomness of OPRF, the probability that this happens is negligible.

Exp. 8 ≈ Exp. 9. The oracle Prim simulates an idealized primitive, and hence, provides uniform outputs by the pseudorandomness of OPRF. Therefore, as long as Prim is not queried on an input, the output cannot be anticipated. In addition, our simulator sampled y_{win} uniformly at random. Therefore, this transition is computationally indistinguishable.

Next, we consider the setting in which the adversary corrupts the dealer. We again proceed in a series of hybrid executions.

Exp. 0. The first experiment is the honest execution of our protocol depicted in Figure 9.

Exp. 1. In Exp. 1, we replace the key generation protocol of Figure 7 using the 2PC simulator $\mathsf{Sim}_{\mathsf{DKG}}$. Such a simulator exists for this protocol, as our theorem assumes it is a secure 2PC protocol. $\mathsf{Sim}_{\mathsf{DKG}}$ reads a key $\mathsf{pk}_{\mathsf{tmp}} \in \mathbb{G}$ and outputs $\mathsf{sk}_{\mathsf{tmp,P}}$ to $\mathsf{Sim}_{\mathsf{D}}$.

Exp. 2. In Exp. 2, we replace the joint adaptor signing protocols of Figure 8 using the 2PC simulator $\mathsf{Sim}_{\mathsf{pSign}}$. Such a simulator exists for this protocol, as our theorem assumes it is a secure 2PC protocol. $\mathsf{Sim}_{\mathsf{pSign}}$ reads a public key $\mathsf{pk}_{\mathsf{tmp}} \in \mathbb{G}$, a secret key $\mathsf{sk}_{\mathsf{tmp,P}}$, and a pre-signature $\widetilde{\sigma}$ and has no output to $\mathsf{Sim}_{\mathsf{D}}$. $\mathsf{Sim}_{\mathsf{pSign}}$ ensures, that the pre-signature computed in the 2PC equals $\widetilde{\sigma}$.

Exp. 3. In Exp. 3, the simulator computes the OPRF outputs of all possible guesses $\hat{y_{\mathsf{gss}}} \in [2^{\ell}]$. If none of these outputs matches y_{win}, the simulator aborts with the message abort_0. The simulator stores the winning guess y_{tgt} and y_{win}, which are found if the simulator does not abort.

Exp. 4. In Exp. 4, the simulator replaces $\mathsf{OPRF.Req}(y_{\mathsf{gss}})$ in the claim dealer phase by $\mathsf{OPRF.Req}(y_{\mathsf{tgt}})$. However, only if the functionality returns $b = 1$, it adapts $\widetilde{\sigma}_{\mathsf{DP}}$ with y_{win}.

Exp. 5. In Exp. 5, the simulator aborts with message abort_1, if extraction of z fails in the funding phase, i.e., if the adversary posts some signature σ^* which is valid under $\mathsf{pk}_{\mathsf{tmp}}$, but it holds that $(Z, \mathsf{AS.Extract}(\mathsf{pk}_{\mathsf{tmp}}, \widetilde{\sigma}_{\mathsf{PD}}, \sigma^*, Z)) \notin \mathcal{R}$.

Exp. 6. In Exp. 6, the simulator aborts with message abort_2, if in the claim phase after decrypting ct, the result is not well-formed, i.e., $\mathsf{res} \neq \mathsf{req}^{\mathsf{sk}_\mathsf{D}}$.
Exp. 7. In Exp. 7, the simulator aborts with message abort_3, if adapting the pre-signature $\widetilde{\sigma}_{\mathsf{DP}}$ with y_{win} fails, i.e., it holds that $\Sigma.\mathsf{Vrfy}(\mathsf{pk}_{\mathsf{tmp}}, \mathsf{tx}_{\mathsf{DP}}, \mathsf{AS.Adapt}(Y, \widetilde{\sigma}_{\mathsf{DP}}, y_{\mathsf{win}})) = 0$.
Exp. 8. In Exp. 8, the simulator aborts with message abort_4 if the adversary attempts to post a tuple $(\mathsf{tx}^*, \sigma^*)$ to $\mathbb{B}$, in which σ^* is valid w.r.t. pk_P on tx^*, and $\mathsf{tx}^* \notin \{\mathsf{tx}_\mathsf{P}, \mathsf{tx}_{\mathsf{rfndP}}\}$.
Simulator Sim_P. The execution of the simulator is defined as the execution in Exp. 8. The simulator Sim_P reads as input $(\nu_\mathsf{D}, p, \ell, \mathbf{T}_\mathsf{D}, \mathbf{T}_\mathsf{P})$ and starts by computing a signing keypair $(\mathsf{pk}_\mathsf{P}, \mathsf{sk}_\mathsf{P}) \leftarrow \mathsf{KGen}(\lambda)$. Then, Sim_P internally executes $\mathcal{A}$ on input $(\nu_\mathsf{D}, p, \ell, \mathbf{T}_\mathsf{D}, \mathbf{T}_\mathsf{P}, \mathsf{pk}_\mathsf{P})$. Eventually, $\mathcal{A}$ outputs $(\mathsf{pk}_\mathsf{D}, Y_{\mathsf{win}}, \pi)$.
Setup: Sim_P sends $(\nu_\mathsf{D}, p, \mathsf{pk}_\mathsf{D}, \mathsf{pk}_\mathsf{P}, \mathbf{T}_\mathsf{D}, \mathbf{T}_\mathsf{P})$ to $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$.
Funding:, $\mathcal{A}$ eventually attempts to run the DKG protocol. To run this protocol, Sim_P first samples a keypair $(\mathsf{pk}_{\mathsf{tmp}}, \mathsf{sk}_{\mathsf{tmp}}) \leftarrow \mathsf{KeyGen}(\lambda)$ and interacts with $\mathcal{A}$ using $\mathsf{Sim}_{\mathsf{DKG}}$ on input $\mathsf{pk}_{\mathsf{tmp}}$. Eventually, $\mathcal{A}$ receives back $\mathsf{sk}_{\mathsf{tmp,D}}$. After running this DKG, the adversary eventually attempts to post $(\mathsf{tx}_\mathsf{D}, \sigma_\mathsf{D})$ on-chain. The simulator computes a signature σ_P and provides the pair $(\mathsf{tx}_\mathsf{P}, \sigma_\mathsf{P})$ to $\mathcal{A}$. In addition, the simulator invokes the `Freeze` interface of $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$.
Claim Dealer: The simulator extracts $(y'_{\mathsf{tgt}}, \mathsf{sk}'_\mathsf{D})$ using the proof system's extractor. If this extraction does not yield a valid secret key for pk_D, or $Y_{\mathsf{win}} \neq g^{\mathsf{OPRF.Eval}(\mathsf{sk}'_\mathsf{D}, y'_{\mathsf{tgt}})}$, the simulator aborts with the message abort_0. The simulator stores y_{win}, which is found if the simulator does not abort. Then, the simulator computes the request $\mathsf{req} \leftarrow \mathsf{OPRF.Req}(y_{\mathsf{tgt}})$. and sends req to $\mathcal{A}$, which eventually responds with the tuple $(Z, ct_\mathsf{D}, \pi_\mathsf{D})$ to $\mathcal{A}$. To run both pre-signing protocols, Sim_P locally pre-signs each $\mathsf{tx}_{\mathsf{PD}}, \mathsf{tx}_{\mathsf{DP}}$ using $\mathsf{sk}_{\mathsf{tmp}}$, resulting in $\widetilde{\sigma}_{\mathsf{PD}}$, $\widetilde{\sigma}_{\mathsf{DP}}$, and uses $\mathsf{Sim}_{\mathsf{pSign}}$ on input $(\mathsf{sk}_{\mathsf{tmp,P}}, \widetilde{\sigma}_{\mathsf{PD}})$ and $(\mathsf{sk}_{\mathsf{tmp,P}}, \widetilde{\sigma}_{\mathsf{DP}})$, to enforce these pre-signatures as outputs of the pre-signing protocols. Eventually, $\mathcal{A}$ posts $(\mathsf{tx}_{\mathsf{PD}}, \sigma_{\mathsf{PD}})$ to $\mathbb{B}$. If this happens, the simulator calls the `ClaimDealer` interface of $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$, which responds with a bit b.
Claim Party: If $b = 1$, the simulator adapts $\widetilde{\sigma}_{\mathsf{DP}}$ with y_{win} into σ_{DP} and calls the `ClaimParty` interface of $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$.
Timeout: If funding was successful, but $\mathcal{A}$ did not participate in the protocol until $\mathbf{T}_\mathsf{P}$ is up, Sim_P signs $\mathsf{tx}_{\mathsf{rfndP}}$, sends abort to $\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}$, and forwards $(\mathsf{tx}_{\mathsf{rfndP}}, \sigma_{\mathsf{rfndP}})$ to $\mathcal{A}$.

On top of this simulation, Sim_P aborts if the adversary forges signatures under pk_P (c.f., Exp. 8) or $\mathsf{pk}_{\mathsf{tmp}}$ (c.f., Exp. 5), or computes the encrypted OPRF response unfaithfully (c.f., Exp. 6), or adapting the adaptor signature fails (c.f., Exp. 7).

We next show that the differences between two neighboring experiments are computationally indistinguishable. Since we have only polynomially many experiments, this establishes the first part of our theorem, since by this negligible transition, we have

$$\mathtt{IDEAL}_{\mathcal{F}^{\mathbb{B}}_{\mathsf{ProSwap}}, \mathsf{Sim}_\mathsf{P}} \approx \mathtt{REAL}_{\Pi^{\mathbb{B}}_{\mathsf{ProSwap}}, \mathcal{A}_\mathsf{D}}.$$

Exp. 0 $\approx$ Exp. 1. This transition is analogous to Exp. 1$\approx$Exp. 2 in the simulation of the dealer: the honest DKG protocol is replaced by the 2PC simulator $\mathsf{Sim}_{\mathsf{DKG}}$, and indistinguishability follows from the security of Γ_{KG}.
Exp. 1 $\approx$ Exp. 2. This transition is analogous to Exp. 2$\approx$Exp. 3 in the simulation of the dealer: the honest pre-signing protocol is replaced by the 2PC simulator $\mathsf{Sim}_{\mathsf{pSign}}$, and indistinguishability follows from the security of Γ_{pSign}.
Exp. 2 $\approx$ Exp. 3. This transition is negligible by the knowledge soundness of the proof system Π. The experiments differ only when witness extraction fails, violating knowledge soundness.
Exp. 3 $\approx$ Exp. 4. The indistinguishability between these experiments follows from the request privacy of OPRF, and we build a reduction to show this. The challenger for request privacy provides two interfaces Req and Finalize to our simulator. Instead of computing the request as in Exp. 3, the simulator calls $\mathsf{Req}(\mathsf{pk}_\mathsf{D}, y_{\mathsf{gss}}, y_{\mathsf{tgt}})$, and receives back two requests. It forwards the first request to $\mathcal{A}$ during the claim dealer phase. The rest of Exp. 3 can be simulated perfectly by the simulator. Eventually, the adversary finishes, and our reduction provides the transcript to the distinguisher, which outputs a bit b'. Our reduction outputs the same bit b' as the guess against the request privacy property of OPRF. If the request privacy challenger

set $b = 0$, then our reduction sent the request for y_{gss}, and hence simulated Exp. 3 perfectly. In the other case, if $b = 1$, it simulated Exp. 4 perfectly. Hence, any ability to distinguish these experiments leads to our reduction breaking the request privacy of OPRF. Therefore, the gap between the experiments is negligible.
Exp. 4 ≈ Exp. 5. Indistinguishability between these experiments follows from the extractability of AS: If there is a gap between the experiments, then the adversary outputs a signature valid under $\mathsf{pk}_{\mathsf{tmp}}$ which does not allow extracting with $\widetilde{\sigma}_{\mathsf{PD}}$. In this situation, we can build a reduction that receives and provides $\mathsf{pk}_{\mathsf{tmp}}$ as input. The simulator computes all pre-signatures using the provided pre-sign oracle, and simulates the experiment as in Exp. 5. When an abort occurs, the simulator outputs the forged signature. By assumption of the abort, this signature forgery breaks extractability, while our reduction is efficient. Therefore, this abort happens only with negligible probability.
Exp. 5 ≈ Exp. 6. Indistinguishability between these experiments follows from the soundness of the proof system Π: If the decryption of the simulator does not equal the well-formed OPRF evaluation, the adversary managed to compute a valid proof for an invalid statement, and the simulator can use this proof to break soundness.
Exp. 6 ≈ Exp. 7. Indistinguishability between these experiments follows canonically from the adaptability of AS.
Exp. 7 ≈ Exp. 8. Indistinguishability between these experiments follows canonically from the unforgeability of Σ.

6 Extensions and Limitations

Cross-chain swaps. Our proposed probabilistic swap definition and protocols rely solely on the signature verification and locking capabilities of the underlying blockchain, similar to their deterministic counterparts [61]. Therefore, our protocol is also applicable to settings in which the dealer uses assets on a blockchain different from the party's. The only difference is that both parties lock the funds in two joint accounts, one on each blockchain. The remainder of the protocol remains unchanged. In case the two chains employ signature schemes across different curves (e.g., Bitcoin and Cardano), we would require additional cross group equality proofs for discrete log statements which are standard in the literature.

Any winning probability. Our protocol currently only supports a winning probability of $1/m$ for $m \in \mathbb{N}$. This is because the party can evaluate a single OPRF per query, and we used a range proof that shows $y_{\mathsf{tgt}} \leq m$. We can slightly modify our protocol to enable any winning probability $a/b \in \mathbb{Q}$. For the denominator, we simply set $m = b$. To modify the numerator from 1 to a, the dealer can allow the party to simultaneously send a requests, which the dealer evaluates and encrypts under the same z. This allows the party to make a guesses per exchange, enhancing the winning probability to a/b, while keeping the on-chain footprint unchanged.

Post-quantum security. Our current instantiation is built on Schnorr signatures and hence relies on the hardness of the discrete logarithm problem. We made this choice to be compatible with Bitcoin and related blockchains at the time of writing. Nevertheless, we conjecture that our protocol can also be instantiated using post-quantum (PQ) secure building blocks. In particular, there exist PQ secure 2Hash-based OPRFs [11] as well as post-quantum adaptor signature schemes [56,19]. Corresponding CPA-secure public-key encryption schemes, compatible in the sense that adaptor signatures can be used to exchange decryption keys, include Regev's LWE-based encryption scheme [53] and isogeny-based PKE [9], respectively. Moreover, PQ proof systems exist that can verify the well-formedness of ciphertexts and OPRF evaluations [31].

Avoiding on-chain timelocks. One on-chain difference between our probabilistic swap protocol and the deterministic counterpart of [61] is that, in addition to requiring on-chain signature verification, our protocol also requires on-chain coin locking. We added this to our ideal functionality, since most deployed swap protocols rely on this kind of coin-locking. However, we can also use verifiable timed signatures (VTS) from [61] to realize this without using on-chain scripts. This changes the on-chain footprint of our protocol to just ordinary transactions without any scripting. The security of our protocol still holds, but Theorem 1 would require the VTS security on top.

Deployment on the Lightning network. Our protocol can also be slightly modified to run on PTLC-based layer2 networks, such as the Lightning Network, once PTLCs become active. To do this, the dealer and party run the funding phase only once, locking the funds into the payment channel. Then, they can run the protocol several times by just executing the claim dealer and claim party protocol subparts. Running these subparts reveals the adaptor witnesses needed to settle the corresponding channel updates. Once enough probabilistic exchanges have happened, both parties can close the channel. This results in an on-chain footprint identical to the footprint of an ordinary payment channel. While PTLCs are not yet deployed for the Lightning Network, they are actively under development [12], so we expect them to be available in the next few years. For today's HTLC-based Lightning interface, we can bind the claim dealer and claim party protocols to the release of hash-preimage, instead of adaptor signatures using off-chain link proofs. We demonstrate the feasibility of this in a variant of our prototype as described in Section 7.

Difficulty of two-sided probabilistic swaps. We provide intuition why protocols in which both parties obtain probabilistic outputs cannot be realized using minimal blockchain functionality, following the impossibility result of [7]. We model the parties as interactive machines that sequentially execute a probabilistic swap protocol. At the start of the protocol, neither party knows whether it will receive its payout; otherwise, a party facing an unfavorable outcome would not participate. At the end of the protocol, however, both parties learn their outcomes. Therefore, for each party, there must exist a *first* message after which it can determine whether it will receive its payout; following [7], we call such a message *deciding*. Consider the deciding message for party j. Upon receiving it, party j learns its partial outcome from the protocol and, if it is unfavorable, party j has no incentive to continue and can simply abort. Crucially, in any sequential protocol on a scriptless blockchain, the deciding messages for the two parties cannot arrive simultaneously [7]. One party must receive its deciding message strictly before the other. At that point, if the outcome is unfavorable, that party can abort before the other party receives its own deciding message. This creates an inherent asymmetry: the aborting party can condition its behavior on information that the other party does not yet have. This argument shows that, in the absence of additional trust assumptions or stronger on-chain functionality such as a randomness beacon [16,40] or enforceable execution [48,8], two-sided probabilistic swaps cannot be realized. In contrast, our asymmetric formulation avoids this impossibility by ensuring that only one party's outcome is probabilistic, and hence that only one deciding message is required.

7 Prototype and Benchmarks

We implement an ℓ-bit instantiation of our probabilistic swap protocol in Rust, giving winning probability $p = 2^{-\ell}$, and evaluate it on both the Bitcoin and Litecoin testnets. The artifact implements the protocol equations, transcripts, OPRF-based claim mechanism, and transaction flowsand is available at [57]. We implement both instantiations of the well-formedness proof for Y_{win}: Bulletproofs over an R1CS constraint system using the constraint-friendly map-to-elliptic-curve-group relation of Groth et al. [33] and a MiMC-based hash-to-scalar, and our cut-and-choose construction with $\lambda = 80$. We instantiate the adaptor signature scheme with Schnorr signatures over the secp256k1 curve, matching the cryptographic primitives natively supported by Bitcoin and Litecoin Taproot. The implementation uses the Rust crates `k256` for secp256k1 arithmetic, `ark-bulletproofs` and `merlin` for the Bulletproof backend and transcripts, and `bitcoin`/`bitcoincore-rpc` for Taproot transaction construction and daemon integration. The OPRF follows the 2HashDH construction from [38]. All benchmarks were obtained on a M3 Max MacBook Pro with 128GB memory.

Micro benchmarks. We evaluated the local computation time of each aspect of our protocol and show benchmarks for the proof of well-formedness of Y_{win} in Table 2. As expected, all other local cryptographic operations are sub-millisecond, as shown in Table 3.

For the well-formedness proof, cut-and-choose is faster for proving at small ℓ, with proof size and running time growing exponentially in ℓ. At $\ell = 10$, it proves in 0.27s and verifies in 0.25s, with a 296KB proof. At $\ell = 16$, the proof grows to roughly 16MB and takes roughly 16s to prove and 15s to verify. The Bulletproof instantiation has a relatively constant proving time of 5.6s, verification time of 0.35s, and proof size of

2.8KB. We benchmarked the Bulletproof-based instantiation up to $\ell = 128$, which still falls within the same range. These measurements were expected, since the cut-and-choose approach has to evaluate a Schnorr OR proof over all 2^ℓ elements, so it scales linearly in the winning probability, whereas the Bulletproof-based approach runs a rangeproof, and an R1CS proof of correct hash-to-group, group exponentiation, and hash-to-scalar, which all require constant overhead (or logarithmic for the range-proof). In practice, the choice of instantiation thus depends on the target winning probability and on whether the limiting resource is prover time or proof size. For large ℓ (i.e., small winning probability $p \ll 1$), the Bulletproof instantiation is preferable. For small ℓ (e.g., $p \geq 1/4096$), the cut-and-choose construction provides a sound and faster-proving alternative, while Bulletproof verification and proof size remain nearly constant.

Table 2. Benchmark results for well-formedness proofs at selected values of ℓ ($p = 1/2^\ell$). We benchmark both a vanilla cut-and-choose, and a version with batched Schnorr proofs.

	Bulletproofs			**Cut-and-Choose**			**Batched C&C**		
ℓ	Prove (s)	Verify (s)	Size (KB)	Prove (s)	Verify (s)	Size (KB)	Prove (s)	Verify (s)	Size (KB)
1	5.64	0.347	2.8	0.020	0.019	42.2	0.013	0.009	27.8
4	5.63	0.347	2.8	0.024	0.022	45.7	0.016	0.012	31.4
8	5.62	0.346	2.8	0.083	0.075	105.1	0.076	0.065	90.8
10	5.58	0.346	2.8	0.272	0.246	295.7	0.265	0.235	281.4
12	5.65	0.347	2.8	1.029	0.926	1057.9	1.021	0.916	1043.3
16	5.54	0.347	2.8	16.19	14.53	16,297	16.17	14.53	16,284
128	5.65	0.348	2.8	—	—	—	—	—	—

Lightning HTLC variant. We also implemented a local two-node Lightning prototype using `ldk-node`. This prototype targets the standard HTLC interface exposed by deployed Lightning implementations, and is therefore not the PTLC/adaptor-signature channel construction discussed in Section 6. Since a standard Lightning payment reveals a SHA256 preimage rather than an adaptor-signature witness, we bind each payment hash to the corresponding protocol witness with an off-chain Groth16 proof for the relation

$$\{((X, h), \mathbb{w}) : X = g^{\mathbb{w}} \wedge h = \mathrm{SHA256}(\mathbb{w})\}.$$

We use this relation with $X = Z$ for the dealer payment and with $X = Y_{\mathsf{win}}$ for the prize payment. In our benchmark run, the dealer precomputed a single Y_{win} and three fresh ticket witnesses z_i, opened one local Lightning channel, and executed three HTLC rounds against the same Y_{win}; the first two rounds were losing guesses and the final round was winning. The Groth16 link proofs were generated before channel funding. This precomputation took 21.96s, channel setup took 0.35s, and the online HTLC execution took 1.30s total, or roughly 0.43s per round. Each Groth16 link proof is 128 bytes, with a 65 byte public statement consisting of the compressed secp256k1 point X and Lightning payment hash h.

On-chain cost and fees. A complete winning probabilistic swap execution involves exactly four on-chain transactions: two funding transactions (one per party), one payment-claim transaction by the dealer, and one reward-claim transaction by the party. Each transaction uses a standard locking script $\Lambda(\mathsf{pk}_{\mathsf{tmp}}, T, \mathsf{pk})$ as defined in Figure 2, which requires only a single signature for the claim path and a timelock check for the refund path. These scripts are already supported natively by Bitcoin and Litecoin Taproot via `OP_CHECKSIG` and `OP_CHECKLOCKTIMEVERIFY` (BIP 65), and do not require any additional opcodes or custom spending conditions beyond what standard atomic swaps already use [61]. Consequently, probabilistic swap transactions have the same on-chain structure as ordinary script-path Taproot spends.

In our public Bitcoin testnet4 run, the Bitcoin reward funding transaction paid a fee of 1550 sat and had weight 616WU, while the Bitcoin reward claim paid a fee of 500 sat and had weight 545WU.

Table 3. Microbenchmarks for protocol functions outside $\mathcal{R}_{\mathsf{win}}$. Times are in milliseconds.

Function	Time	Function	Time	Function	Time
$\Gamma_{\mathsf{DL}}.\mathcal{P}$	0.060	OPRF.Eval	0.118	Γ_{pSign}	0.120
$\Gamma_{\mathsf{DL}}.\mathcal{V}$	0.103	Π_{Enc}.Enc	0.091	pVrfy	0.145
OPRF.Req	0.064	Π_{Enc}.Dec	0.050	Adapt	0.287
OPRF.BlindEval	0.043	$\Pi.\mathcal{P}(\mathcal{R}_{\mathsf{Enc}})$	0.300	Extract	0.043
OPRF.Finalize	0.109	$\Pi.\mathcal{V}(\mathcal{R}_{\mathsf{Enc}})$	0.359	Vrfy	0.101

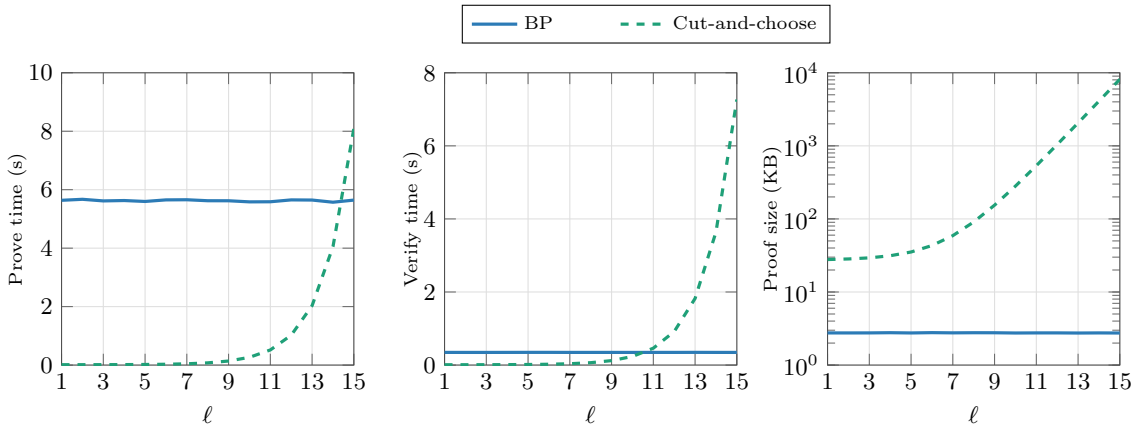


Fig. 10. Performance and proof size for different instantiations of the well-formedness proof of Y_{win}. The parameter ℓ corresponds to the winning party's probability via $p = 2^{-\ell}$.

Cross-chain swap: Bitcoin × Litecoin. To test the cross-chain variant from Section 6, we ran the protocol on Bitcoin testnet4 and Litecoin testnet. Both chains support Taproot Schnorr spends over secp256k1, so the same adaptor-signature code applies on both sides. The public runs used $\ell = 8$, hence winning probability 1/256, and the batched cut-and-choose proof with $\lambda = 80$. In the winning run, the dealer locked the Bitcoin reward and the party locked the Litecoin payment. The dealer claimed the Litecoin payment by revealing the adaptor witness z; the party extracted z, finalized the OPRF, and claimed the Bitcoin reward using y_{win}.

We repeated the experiment with a losing guess on the same public test networks. The dealer still claimed the Litecoin payment, which revealed z. The party extracted z and finalized the OPRF, but the result was not the winning adaptor witness. Adapting the Bitcoin reward signature therefore failed, and the dealer later spent the Bitcoin refund path after the timeout.

Table 4. On-chain transaction IDs for Bitcoin × Litecoin cross-chain probabilistic swaps.

Run	Event	Chain	Transaction ID
Win	Dealer funding	Bitcoin testnet4	`c69fc5c1...fedcec33`
Win	Party funding	Litecoin testnet	`32aa28fc...3fc4f32e`
Win	Dealer claim	Litecoin testnet	`e71cae45...740e96c04f`
Win	Party claim	Bitcoin testnet4	`7ad17bae...127e0fff8`
Lose	Dealer funding	Bitcoin testnet4	`0f42a6e6...aab1ffd1`
Lose	Party funding	Litecoin testnet	`08c9e270...0381fbde`
Lose	Dealer claim	Litecoin testnet	`43a5c864...2cf78e6d`
Lose	Dealer refund	Bitcoin testnet4	`6014382a...06955d89`

Acknowledgements

Paul Gerhart's research has been supported by the Google PhD Fellowship in Privacy, Safety, and Security.

References

1. Emulating op_rand in bitcoin. https://delvingbitcoin.org/t/emulating-op-rand/1409/5 (2025), delving Bitcoin forum discussion
2. Agrawal, S., Miao, P., Mohassel, P., Mukherjee, P.: PASTA: PASsword-based threshold authentication. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 2042–2059. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243839
3. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on Bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 443–458. IEEE Computer Society Press (May 2014). https://doi.org/10.1109/SP.2014.35
4. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostáková, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized channels from limited blockchain scripts and adaptor signatures. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part II. LNCS, vol. 13091, pp. 635–664. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92075-3_22
5. Aumayr, L., Thyagarajan, S.A.K., Malavolta, G., Moreno-Sanchez, P., Maffei, M.: Sleepy channels: Bi-directional payment channels without watchtowers. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 179–192. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3559370
6. Badertscher, C., Campanelli, M., Ciampi, M., Russo, L., Siniscalchi, L.: Universally composable SNARKs with transparent setup without programmable random oracle. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part VII. LNCS, vol. 16006, pp. 225–258. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01907-3_8
7. Baecker, R., Gerhart, P., Katz, J., Schröder, D.: Fair exchange for decentralized autonomous organizations via threshold adaptor signatures. Cryptology ePrint Archive, Report 2025/388 (2025), https://eprint.iacr.org/2025/388
8. Bartoletti, M., Zunino, R.: Constant-deposit multiparty lotteries on bitcoin. In: Financial Cryptography and Data Security. pp. 231–247. Springer International Publishing, Cham (2017)
9. Basso, A., Maino, L.: POKÉ: A compact and efficient PKE from higher-dimensional isogenies. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part II. LNCS, vol. 15602, pp. 94–123. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91124-8_4
10. Bentov, I., Kumaresan, R.: How to use Bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 421–439. Springer, Berlin, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44381-1_24
11. Beullens, W., Dodgson, L., Faller, S.H., Hesse, J.: The 2Hash OPRF framework and efficient post-quantum instantiations. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part VIII. LNCS, vol. 15608, pp. 332–362. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91101-9_12
12. Bitcoin Optech: Point time-locked contracts (ptlcs). https://bitcoinops.org/en/topics/ptlc/, accessed: 2026-04-27
13. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: 20th ACM STOC. pp. 103–112. ACM Press (May 1988). https://doi.org/10.1145/62212.62222
14. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00020
15. Camenisch, J., Lehmann, A., Neven, G.: Optimal distributed password verification. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 182–194. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813722
16. Choudhuri, A.R., Green, M., Jain, A., Kaptchuk, G., Miers, I.: Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 719–728. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134092
17. Davies, G.T., Faller, S.H., Gellert, K., Handirk, T., Hesse, J., Horváth, M., Jager, T.: Security analysis of the WhatsApp end-to-end encrypted backup protocol. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part IV. LNCS, vol. 14084, pp. 330–361. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38551-3_11

18. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 566–598. Springer, Berlin, Heidelberg (Aug 2001). https://doi.org/10.1007/3-540-44647-8_33
19. Esgin, M.F., Ersoy, O., Erkin, Z.: Post-quantum adaptor signatures and payment channel networks. In: Chen, L., Li, N., Liang, K., Schneider, S.A. (eds.) ESORICS 2020, Part II. LNCS, vol. 12309, pp. 378–397. Springer, Cham (Sep 2020). https://doi.org/10.1007/978-3-030-59013-0_19
20. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM **28**(6), 637–647 (Jun 1985). https://doi.org/10.1145/3812.3818, https://doi.org/10.1145/3812.3818
21. Everspaugh, A., Chatterjee, R., Scott, S., Juels, A., Ristenpart, T.: The pythia PRF service. In: Jung, J., Holz, T. (eds.) USENIX Security 2015. pp. 547–562. USENIX Association (Aug 2015), https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/everspaugh
22. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the Fiat-Shamir transform. In: Galbraith, S.D., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Berlin, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34931-7_5
23. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Berlin, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12
24. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Berlin, Heidelberg (Feb 2005). https://doi.org/10.1007/978-3-540-30576-7_17
25. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 295–310. Springer, Berlin, Heidelberg (May 1999). https://doi.org/10.1007/3-540-48910-X_21
26. Gerhart, P., Calsi, D.L., Russo, L., Schröder, D.: Fully-adaptive two-round threshold schnorr signatures from DDH. In: Advances in Cryptology – EUROCRYPT 2026. Lecture Notes in Computer Science, Springer (2026)
27. Gerhart, P., Rausch, D., Schröder, D.: Universally composable adaptor signatures. In: Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery (2026)
28. Gerhart, P., Schröder, D., Soni, P., Thyagarajan, S.A.K.: Foundations of adaptor signatures. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part II. LNCS, vol. 14652, pp. 161–189. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58723-8_6
29. Glaeser, N., Maffei, M., Malavolta, G., Moreno-Sanchez, P., Tairi, E., Thyagarajan, S.A.K.: Foundations of coin mixing services. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 1259–1273. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560637
30. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: 17th ACM STOC. pp. 291–304. ACM Press (May 1985). https://doi.org/10.1145/22145.22178
31. Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part II. LNCS, vol. 14082, pp. 193–226. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38545-2_7
32. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 581–612. Springer, Cham (Aug 2017). https://doi.org/10.1007/978-3-319-63715-0_20
33. Groth, J., Malvai, H., Miller, A., Zhang, Y.N.: Constraint-friendly map-to-elliptic-curve-group relations and their applications. In: Hanaoka, G., Yang, B.Y. (eds.) ASIACRYPT 2025, Part II. LNCS, vol. 16246, pp. 511–543. Springer, Singapore (Dec 2025). https://doi.org/10.1007/978-981-95-5096-8_16
34. Hanzlik, L., Loss, J., Thyagarajan, S.A.K., Wagner, B.: Sweep-UC: Swapping coins privately. In: 2024 IEEE Symposium on Security and Privacy. pp. 3822–3839. IEEE Computer Society Press (May 2024). https://doi.org/10.1109/SP54263.2024.00081
35. Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., Goldberg, S.: TumbleBit: An untrusted Bitcoin-compatible anonymous payment hub. In: NDSS 2017. The Internet Society (Feb / Mar 2017). https://doi.org/10.14722/ndss.2017.23086
36. Hesse, J., Jarecki, S., Krawczyk, H., Wood, C.: Password-authenticated TLS via OPAQUE and post-handshake authentication. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 98–127. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4_4
37. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 233–253. Springer, Berlin, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45608-8_13

38. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 276–291 (2016). https://doi.org/10.1109/EuroSP.2016.30
39. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 456–486. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_15
40. Kaptchuk, G., Green, M., Miers, I.: Giving state to the stateless: Augmenting trustworthy computation with ledgers. In: NDSS 2019. The Internet Society (Feb 2019). https://doi.org/10.14722/ndss.2019.23060
41. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman and Hall, CRC Press, third edn. (2014)
42. Kumaresan, R., Moran, T., Bentov, I.: How to use Bitcoin to play decentralized poker. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 195–206. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813712
43. Kurbatov, O.: Emulating op_rand in bitcoin (2025), https://arxiv.org/abs/2501.16451
44. Lindell, Y.: How to simulate it - A tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046 (2016), https://eprint.iacr.org/2016/046
45. Lindell, Y., Pinkas, B.: Secure two-party computation via cut-and-choose oblivious transfer. Journal of Cryptology **25**(4), 680–722 (Oct 2012). https://doi.org/10.1007/s00145-011-9107-0
46. Madathil, V., Thyagarajan, S.A.K., Vasilopoulos, D., Fournier, L., Malavolta, G., Moreno-Sanchez, P.: Cryptographic oracle-based conditional payments. In: NDSS 2023. The Internet Society (Feb 2023)
47. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: NDSS 2019. The Internet Society (Feb 2019). https://doi.org/10.14722/ndss.2019.23330
48. Miller, A., Bentov, I.: Zero-collateral lotteries in bitcoin and ethereum. In: 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 4–13 (2017). https://doi.org/10.1109/EuroSPW.2017.44
49. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., McCorry, P.: Sprites and state channels: Payment networks that go faster than lightning. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 508–526. Springer, Cham (Feb 2019). https://doi.org/10.1007/978-3-030-32101-7_30
50. Nolan, T.: Alt chains and atomic transfers. https://bitcointalk.org/index.php?topic=193281.0 (2013)
51. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE Computer Society Press (May 2013). https://doi.org/10.1109/SP.2013.47
52. Qin, X., Pan, S., Mirzaei, A., Sui, Z., Ersoy, O., Sakzad, A., Esgin, M.F., Liu, J.K., Yu, J., Yuen, T.H.: BlindHub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts. In: 2023 IEEE Symposium on Security and Privacy. pp. 2462–2480. IEEE Computer Society Press (May 2023). https://doi.org/10.1109/SP46215.2023.10179427
53. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6) (Sep 2009). https://doi.org/10.1145/1568318.1568324, https://doi.org/10.1145/1568318.1568324
54. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: 40th FOCS. pp. 543–553. IEEE Computer Society Press (Oct 1999). https://doi.org/10.1109/SFFCS.1999.814628
55. Tairi, E., Moreno-Sanchez, P., Maffei, M.: A^2L: Anonymous atomic locks for scalability in payment channel hubs. In: 2021 IEEE Symposium on Security and Privacy. pp. 1834–1851. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00111
56. Tairi, E., Moreno-Sanchez, P., Maffei, M.: Post-quantum adaptor signature for privacy-preserving off-chain payments. In: Borisov, N., Díaz, C. (eds.) FC 2021, Part II. LNCS, vol. 12675, pp. 131–150. Springer, Berlin, Heidelberg (Mar 2021). https://doi.org/10.1007/978-3-662-64331-0_7
57. Taylor, J.: Bitcoin probabilistic swaps. https://github.com/EzePze/bitcoin-probabilistic-swaps (2026), accessed: 2026-05-05
58. Taylor, J., Gerhart, P., Thyagarajan, S.A.: How to make delegated payments on bitcoin: A question for the AI agentic future. Financial Cryptography and Data Security (2026), https://fc26.ifca.ai/preproceedings/113.pdf
59. Thyagarajan, S.A.K., Bhat, A., Malavolta, G., Döttling, N., Kate, A., Schröder, D.: Verifiable timed signatures made practical. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1733–1750. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417263
60. Thyagarajan, S.A.K., Malavolta, G.: Lockable signatures for blockchains: Scriptless scripts for all signatures. 2021 IEEE Symposium on Security and Privacy (SP) pp. 937–954 (2021), https://api.semanticscholar.org/CorpusID:231732350
61. Thyagarajan, S.A.K., Malavolta, G., Moreno-Sanchez, P.: Universal atomic swaps: Secure exchange of coins across all blockchains. In: 2022 IEEE Symposium on Security and Privacy. pp. 1299–1316. IEEE Computer Society Press (May 2022). https://doi.org/10.1109/SP46214.2022.9833731

62. Tyagi, N., Celi, S., Ristenpart, T., Sullivan, N., Tessaro, S., Wood, C.A.: A fast and simple partially oblivious PRF, with applications. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 674–705. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-07085-3_23
63. Vanjani, N., Soni, P., Thyagarajan, S.A.K.: Functional adaptor signatures: Beyond all-or-nothing blockchain-based payments. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) ACM CCS 2024. pp. 1493–1507. ACM Press (Oct 2024). https://doi.org/10.1145/3658644.3690240

A Preliminaries

We recall preliminaries on oblivious PRFs, adaptor signatures, and zero-knowledge proofs.

Oblivious PRFs. We recall the security properties of OPRFs: pseudorandomness, unlinkability, and uniqueness. We base our definitions on the respective definitions for partially-oblivious PRFs from [62]. These definitions imply the same notions for OPRFs, since an OPRF is a POPRF, simply by allowing just a single tag $t = \bot$. We provide the security experiments in Figures 11 and 12 and refer to [62] for high-level explanations and the full-fledged definitions. We include uniqueness here, but do not require it in Theorem 1, since in our construction, well-formedness of the encrypted request is proven using an NIZK, which implies uniqueness of the decrypted value.

$\mathsf{PoPRIV}^{b}_{\mathcal{A},\mathsf{OPRF}}(\lambda)$

$pp \leftarrow \mathsf{OPRF.Setup}(\lambda)$
$i \leftarrow 0$
$b' \leftarrow \mathcal{A}^{\mathsf{Req},\mathsf{Finalize}}(pp)$
return b'

$\mathsf{Req}(\mathsf{pk}, x_0, x_1)$

$i \leftarrow i + 1$
$(\mathsf{st}_{i,0}, \mathsf{req}_0) \leftarrow \mathsf{Req}(\mathsf{pk}, x_0)$
$(\mathsf{st}_{i,1}, \mathsf{req}_1) \leftarrow \mathsf{Req}(\mathsf{pk}, x_1)$
return $(\mathsf{req}_b, \mathsf{req}_{1-b})$

$\mathsf{Finalize}(j, \mathsf{res}, \mathsf{res}')$

assert $j \leq i$
$y_b \leftarrow \mathsf{Finalize}(\mathsf{st}_{j,b}, \mathsf{res})$
$y_{1-b} \leftarrow \mathsf{Finalize}(\mathsf{st}_{j,1-b}, \mathsf{res}')$
if $y_0 = \bot \vee y_1 = \bot$
  return $\bot$
return (y_0, y_1)

$\mathsf{UNIQUE}_{\mathcal{A},\mathsf{OPRF}}(\lambda)$

$\mathcal{Q} \leftarrow \emptyset$; $q \leftarrow 0$
$pp \leftarrow \mathsf{OPRF.Setup}(\lambda)$
$(i, j, \mathsf{res}_i, \mathsf{res}_j) \leftarrow \mathcal{A}^{\mathsf{Req}}(pp)$
assert $1 \leq i \leq j \leq q$
$(\mathsf{st}_i, \mathsf{pk}_i, x_i) \leftarrow \mathcal{Q}[i]$
$(\mathsf{st}_j, \mathsf{pk}_j, x_j) \leftarrow \mathcal{Q}[j]$
assert $(\mathsf{pk}_i, x_i) = (\mathsf{pk}_j, x_j)$
$y_i \leftarrow \mathsf{Finalize}(\mathsf{st}_i, \mathsf{res}_i)$
$y_j \leftarrow \mathsf{Finalize}(\mathsf{st}_j, \mathsf{res}_j)$
if $y_0 = \bot \vee y_1 = \bot$
  return 0
return $y_i \neq y_j$

$\mathsf{Req}(\mathsf{pk}, x)$

$q \leftarrow q + 1$
$(\mathsf{st}, \mathsf{req}) \leftarrow \mathsf{Req}(\mathsf{pk}, x)$
$\mathcal{Q}[q] \leftarrow (\mathsf{st}, \mathsf{pk}, x)$
return req

Fig. 11. Request privacy, uniqueness, and pseudorandomness experiments for OPRFs.

For the security proof of our theorem, we require non-blackbox access to the underlying structure of the OPRF. In particular, our simulator needs to be able to program the OPRF output to a target value. To allow this, we restrict our theorem to the family of two-hash OPRFs, which all share the same structure. We follow the abstraction from [11].

Definition 1 (Two Hash OPRF). *We say that an* OPRF *is* two-hash *if it can be written as* $\mathsf{OPRF}(\mathsf{sk}, x) := H_2\big(F(\mathsf{sk}, H_1(x))\big)$*, where* H_1 *and* H_2 *are hash functions, and* F *can be computed efficiently.*

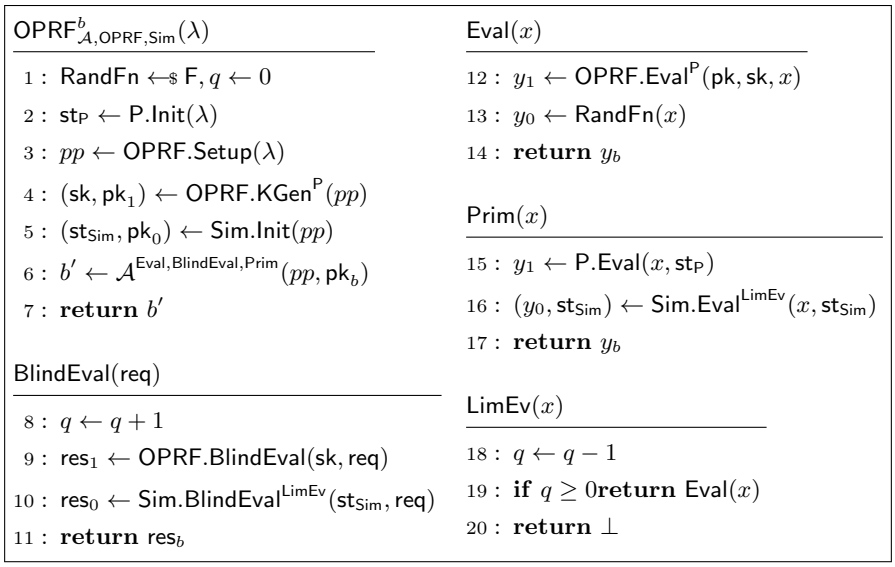


Fig. 12. Pseudorandomness experiment for OPRFs.

Our modified 2HashDH OPRF. To enable cut-and-choose proofs, we slightly modify the 2HashDH OPRF construction, such that the dealer can provide openings *without* revealing the OPRF evaluation's message. In particular, our OPRF has the following form: Let $\mathbb{G}$ be a cyclic group of prime order p, and let H_p and $\mathsf{H}_\mathbb{G}$ be hash functions mapping to $\mathbb{Z}_p$ and $\mathbb{G}$, respectively. The public key of the server has the form $\mathsf{pk} = \big(X = g^{\mathsf{sk}}, \{g^{\alpha_i}\}, \{g^{\alpha_j}\}, \{\alpha_i\}\big)$.

To form a request for input x, the client runs $\mathsf{Req}(x)$ by sampling $r \leftarrow\!\$\ \mathbb{Z}_p$ and computing $\mathsf{req} = \mathsf{H}_\mathbb{G}(x)^r$. Upon receiving req, the server computes $\mathsf{res} = \mathsf{BlindEval}(\mathsf{sk}, \mathsf{req}) = \mathsf{req}^{\mathsf{sk}}$. Requests and responses are identical to the original 2HashDH [38]. However, to finalize, the client runs $\mathsf{Finalize}(x, r, \mathsf{res})$ to remove the blinding and obtains $\mathsf{Eval}(\mathsf{sk}, x) = \{\mathsf{H}_p(\mathsf{pk}, \mathsf{res}^{\alpha_i/r})\} = \{\mathsf{H}_p(\mathsf{pk}, \mathsf{H}_\mathbb{G}(x)^{\mathsf{sk}\cdot\alpha_i})\}$ for all elements α_i part of the public key. The main difference between this OPRF and 2HashDH is two things: First, the first input to H_p, is pk in our case and x in the case of 2HashDH. The second difference is that the OPRF finalization is run on each partial public key element, instead of a single one.

This change does not change the security of the OPRF. To see this, we first recall the pseudorandomness proof of 2HashDH. The proof of pseudorandomness of 2HashDH is based on the one-more-gap DH assumption [38]. The simulator handles queries to the RO $\mathsf{H}_\mathbb{G}$ oracle using the challenge oracle of the Gap-OMDH assumption, which outputs CDH challenges; it also sets X to a CDH challenge from the help oracle. When the adversary queries the oracle H_p on input (x, Y), the simulator checks, if Y is a valid CDH solution by querying $(\mathsf{H}_G G(x), \mathsf{pk}, Y)$ to the DDH oracle of the Gap-OMDH assumption. If this is the case, and the quota q of the pseudorandomness experiment is greater than or equal 0, i.e., the adversary did more $\mathsf{BlindEv}$ queries than queries to H_p on such valid inputs, the simulator simply queries the LimEv oracle, and can program the output of H_p accordingly. However, if the adversary comes up with such an input (x, Y) that denotes a solution for CDH, but the quota q is exhausted, the simulator cannot use the LimEv oracle to answer this query. Yet, this is not a problem for the simulator, since in this case the adversary provided one more solution to the CDH problem, which the simulator can output to break the Gap-OMDH assumption. As the Gap-OMDH assumption is hard, this edge case occurs only with negligible probability, and the simulator suffices to show pseudorandomness.

To show the pseudorandomness of our scheme in a similar fashion, we simulate the RO H_p and the $\mathsf{BlindEv}$ oracle exactly as in the proof of 2HashDH. However, when the adversary provides a value (pk, Y) to the H_p oracle, we handle the situation differently. This is, the simulator checks for all pairs $(x, H_p(x))$ the adversary received from H_p so far, and all values $\alpha_i \in \mathsf{pk}$, if the triple $(\mathsf{pk}, H_p(x), Y^{1/\alpha_i})$ denotes a valid

DDH triple. This introduces a runtime overhead of $64 \cdot |mathcalQ|$, where $\mathcal{Q}$ is the size of queries to H_p, but is still efficient. In addition, the domain of the OPRF is just 64x bigger, so for each message x, the simulator partitions the output of the LimEv oracle in 64 parts, and if $(H(x), \alpha_i, pk)$ denotes a valid CDH triple, hands out the $i-$th part of this partition. This resolves the issue that for a single CDH solution, the adversary can query the oracle H_p 64 times. Therefore, whenever the simulator cannot answer a query to the adversary as the LimEv oracle did not provide a solution, the simulator finds a solution for the Gap-OMDH assumption.

Adaptor signatures. We recall the two main security guarantees of adaptor signatures, namely extractability and pre-signature adaptability following the notion of [28].

Definition 2 (Extractability). *An adaptor signature scheme* AS *is extractable, if for every* PPT *adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that for every* $\lambda \in \mathbb{N}$

$$\Pr[\mathsf{Ext}_{\mathcal{A},\mathsf{AS}}(\lambda) = 1] \leq \mathsf{negl}(\lambda) \ ,$$

where the experiment $\mathsf{Ext}_{\mathcal{A},\mathsf{AS}}$ *is described in Figure 13, and the probability is taken over the random choices of all probabilistic algorithms.*

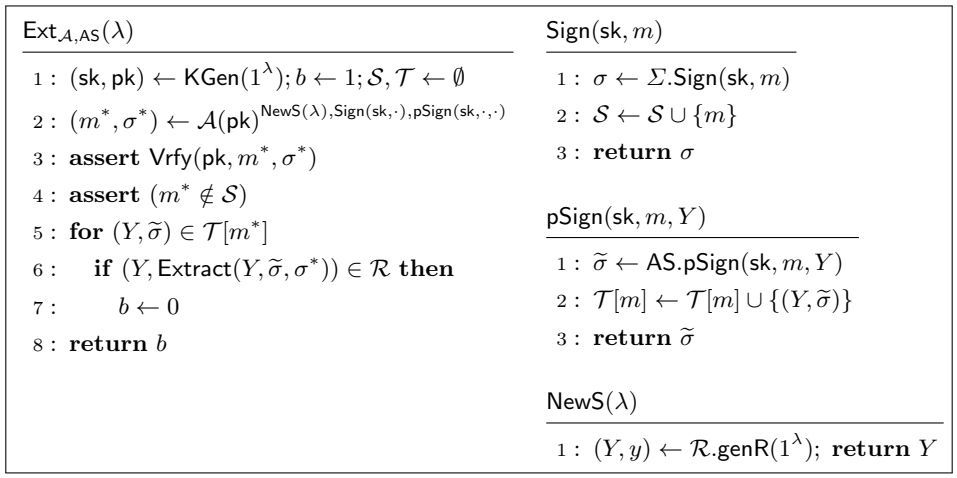


Fig. 13. The security game $\mathsf{Ext}_{\mathcal{A},\mathsf{AS}}(\lambda)$.

Definition 3 (Pre-signature adaptability). *An adaptor signature scheme* AS *satisfies* pre-signature adaptability*, if for all* $\lambda \in \mathbb{N}$*, messages* $m \in \{0,1\}^*$*, statement/witness pairs* $(Y, y) \in \mathcal{R}$*, public keys* pk *and pre-signatures* $\widetilde{\sigma} \in \{0,1\}^*$ *we have* $\mathsf{pVrfy}(\mathsf{pk}, m, Y, \widetilde{\sigma}) = 1$*, then* $\mathsf{Vrfy}(\mathsf{pk}, m, \mathsf{Adapt}(\mathsf{pk}, \widetilde{\sigma}, y)) = 1$.

A.1 Non-interactive Arguments

We follow the notion of [26]. A *non-interactive argument system* (NARG) for relation $\mathcal{R}$ in the random oracle model, denoted by Π, consists of a tuple of algorithms $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ having black-box access to a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$, with the following syntax:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: takes as input the security parameter 1^λ and outputs public parameters pp.
- $\pi \leftarrow \mathcal{P}^{\mathsf{H}}(\mathsf{pp}, \mathbb{x}, \mathbb{w})$: takes as input parameters pp, a statement $\mathbb{x}$ and witness $\mathbb{w}$, and outputs a proof π if $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$.
- $b \leftarrow \mathcal{V}^{\mathsf{H}}(\mathsf{pp}, \mathbb{x}, \pi)$: takes as input parameters pp, a statement $\mathbb{x}$ and proof π, and it accepts ($b = 1$) or rejects ($b = 0$).

In this work, we consider NARGs with *transparent* setup, i.e. pp can be generated with a call to the random oracle. For this reason, we may omit pp in the description of the prover and verifier algorithms as it is implicit.

Definition 4 (Completeness). *A NARG Π satisfies* completeness *if for every $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, it holds that*

$$\Pr\left[\begin{matrix} b = 1 : \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \pi \leftarrow \mathcal{P}^{\mathsf{H}}(\mathsf{pp}, \mathbb{x}, \mathbb{w}); \\ b \leftarrow \mathcal{V}^{\mathsf{H}}(\mathsf{pp}, \mathbb{x}, \pi) \end{matrix}\right] = 1.$$

Definition 5 (Soundness). *A NARG Π satisfies* soundness *if for every PPT adversary $\mathcal{A}$, it holds that*

$$\Pr\left[\begin{matrix} b = 1 \wedge \mathbb{x} \notin \mathcal{L}_{\mathcal{R}} : \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ (\mathbb{x}, \pi) \leftarrow \mathcal{A}^{\mathsf{H}}(\mathsf{pp}); b \leftarrow \mathcal{V}^{\mathsf{H}}(\mathsf{pp}, \mathbb{x}, \pi) \end{matrix}\right]$$

is negligible, where $\mathcal{L}_{\mathcal{R}} \coloneqq \{\mathbb{x} \mid \exists \mathbb{w} : (\mathbb{x}, \mathbb{w}) \in \mathcal{R}\}$ is the set of true-statements.

If soundness holds with respect to a computationally unbounded adversary, we say that the argument has *statistical* (or *information-theoretic*) soundness, and we call the argument a *proof* system.

Zero-knowledge. Informally, an argument is zero-knowledge if a proof reveals no information about the witness [30]. We formalize this property by following the syntax of [22]. A zero-knowledge simulator $\mathcal{S}$ is defined as a stateful algorithm, whose initial state $\mathsf{st} = \mathsf{pp}$, that operates in two modes. The first mode, $(y, \mathsf{st}') \leftarrow \mathcal{S}(1, \mathsf{st}, a)$ takes care of handling calls to the oracle H on input a query a. The second mode, $(\pi, \mathsf{st}') \leftarrow \mathcal{S}(2, \mathsf{st}, \mathbb{x})$ simulates a proof for the input statement $\mathbb{x}$. We define the following *wrapper* oracles, that are stateful and share their internal state:
- $\mathcal{S}_1(a)$ returns the first output of $\mathcal{S}(1, \mathsf{st}, a)$;
- $\mathcal{S}_2(\mathbb{x}, \mathbb{w})$ returns the first output of $\mathcal{S}(2, \mathsf{st}, \mathbb{x})$ if $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ and $\perp$ otherwise;
- $\mathcal{S}'_2(\mathbb{x})$ returns the first output of $\mathcal{S}(2, \mathsf{st}, \mathbb{x})$.

Definition 6 (Zero-knowledge). *A NARG Π is zero-knowledge if there exists a simulator $\mathcal{S}$, with wrapper oracles $\mathcal{S}_1, \mathcal{S}_2$, such that for all PPT adversaries $\mathcal{A}$ it holds that:*

$$\Pr\left[\begin{matrix} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathcal{A}^{\mathcal{P}, \mathsf{H}}(\mathsf{pp}) = 1 \end{matrix}\right] \approx \Pr\left[\begin{matrix} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathcal{A}^{\mathcal{S}_1, \mathcal{S}_2}(\mathsf{pp}) = 1 \end{matrix}\right]$$

Notice that zero-knowledge is a security property that is only guaranteed for valid statements in the language, hence the above definition uses $\mathcal{S}_2$ as a proof simulation oracle. Also, a zero-knowledge NARG with statistical soundness is simply referred to as a non-interactive zero-knowledge proof (NIZK) [13].

Simulation exctractability. A strong security property of NARGs is *simulation extractability* [18,32]. Informally, from a prover producing a valid proof after seeing polynomially many proofs provided by a simulator (even for *false* statements), it is possible to extract a valid witness. This property can be formalized in many different ways, depending on the model and the constraints of the adversary and the extractor. We refer to [6] for a recent work that summarizes the main differences.

Definition 7 (Simulation extractability). *A NARG Π is simulation-extractable with respect to a zero-knowledge simulator $\mathcal{S}$, with wrapper oracles $\mathcal{S}_1, \mathcal{S}'_2$, if for all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}$ such that:*

$$\Pr\left[\begin{matrix} \mathcal{V}^{\mathcal{S}_1}(\mathbb{x}, \pi) = 1 \\ \wedge\ (\mathbb{x}, \pi) \notin \mathcal{Q} \\ \wedge\ (\mathbb{x}, \mathbb{w}) \notin \mathcal{R} \end{matrix} \middle| \begin{matrix} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{x}, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_1, \mathcal{S}'_2}(\mathsf{pp}) \\ \mathbb{w} \leftarrow \mathcal{E}(\mathsf{st}, \mathbb{x}, \pi) \end{matrix}\right] \in \mathsf{negl}(\lambda)$$

where st is the final internal state of $\mathcal{S}$, and $\mathcal{Q}$ is the set of statement-proof pairs obtained by $\mathcal{A}$ when interacting with $\mathcal{S}'_2$.

The previous definition implies the weaker notion of *knowledge soundness* [54]. Informally, a NARG is knowledge-sound if there exists a PPT extractor, but without simulating proofs to the adversary first.

B Instantiation of Proofsystems

Our protocol requires proofs of well-formedness and proofs of knowledge. In this section, we provide instantiations of the relations to be proven, along with sigma protocols for proving them. We keep the sigma protocols interactive, but they can be made non-interactive using the Fiat-Shamir transform [23].

Proof of well-formed winning statement Y_{win}. The relation $\mathcal{R}_{\mathsf{win}}$ is satisfied, if a winning statement Y_{win} is well-formed w.r.t. some OPRF public key pk, and a length parameter ℓ. For this, we provide two relations. The first one, $\mathcal{R}_{\mathsf{win}}$, we use when proving well-formedness using Bulletproofs.

$$\mathcal{R}_{\mathsf{win}} := \left\{ \begin{matrix} \mathbb{x} = (\mathsf{pk}, Y_{\mathsf{win}}, \ell) \\ \mathbb{w} = (\mathsf{sk}, y_{\mathsf{tgt}}) \end{matrix} : \begin{matrix} |y_{\mathsf{tgt}}| \leq 2^{\ell}, \\ y_{\mathsf{win}} = \mathsf{OPRF.Eval}(\mathsf{sk}, y_{\mathsf{tgt}}), \\ Y_{\mathsf{win}} = g^{y_{\mathsf{win}}} \wedge \mathsf{pk} = g^{\mathsf{sk}} \end{matrix} \right\}$$

To prove this relation, we can instantiate the RO and prove well-formedness of $\mathbb{x}$ using, e.g., Bulletproofs [14] and a constraint system like R1CS [51]. If we do not want to instantiate the RO for our proof, we can use the relation $\mathcal{R}'_{\mathsf{win}}$ for the well-formedness of Y_{win}.

$$\mathcal{R}'_{\mathsf{win}} := \left\{ \begin{matrix} \mathbb{x} = (\mathsf{pk}, ct, Y_{\mathsf{win}}, \ell) \\ \mathbb{w} = (\mathsf{sk}, y_{\mathsf{tgt}}) \end{matrix} : \begin{matrix} |y_{\mathsf{tgt}}| \leq 2^{\ell} \\ (Y_{\mathsf{win}}, y_{\mathsf{win}}) \leftarrow \mathsf{genR}(\lambda) \\ \rho = \mathsf{OPRF.Eval}(\mathsf{sk}, y_{\mathsf{tgt}}), \\ ct = \rho + y_{\mathsf{win}} \wedge \mathsf{pk} = g^{\mathsf{sk}} \end{matrix} \right\}$$

To prove well-formedness using $\mathcal{R}'_{\mathsf{win}}$, we can use the cut-and-choose strategy, as depicted in Figures 17 and 18.

Proof of well-formed OPRF response encryption. This proof is the technical cornerstone of the OPRF-as-a-service abstraction introduced in Section 3: it lets the party verify that the dealer has honestly computed a blind OPRF evaluation on the party's request and locked the result in a well-formed ciphertext, without yet revealing the output. The relation $\mathcal{R}_{\mathsf{Enc}}$ is satisfied if a ciphertext ct is well-formed w.r.t. some OPRF public key pk, a statement Z, and an OPRF request req.

$$\mathcal{R}_{\mathsf{Enc}} := \left\{ \begin{matrix} \mathbb{x} = (\mathsf{pk}, Z, \mathsf{req}, ct) \\ \mathbb{w} = (\mathsf{sk}, z) \end{matrix} : \begin{matrix} \mathsf{pk} = g^{\mathsf{sk}} \wedge Z = g^{z}, \\ \mathsf{res} = \mathsf{OPRF.BlindEv}(\mathsf{sk}, \mathsf{req}), \\ ct = \Pi_{\mathsf{Enc}}.\mathsf{Enc}(Z, \mathsf{res}) \end{matrix} \right\}$$

If Π_{Enc} is instantiated with ElGamal encryption, we show this well-formedness using a Chaum-Pedersen sigma protocol, depicted in Figure 14.

Lemma 1. *The sigma protocol in Figure 14 is complete, 2-special sound with knowledge error* $1/p$*, and honest-verifier zero-knowledge.*

Proof. Completeness. For any valid witness (sk, α) the verification equations follow directly from expanding R_1, R_2, R_3 and the responses.

Special soundness. Given two accepting transcripts with identical first message (R_1, R_2, R_3) but distinct challenges $c \neq c'$, let $e = c - c' \neq 0$. From the first equation $g^{z_{\mathsf{sk}} - z'_{\mathsf{sk}}} = \mathsf{pk}^{e}$, so $\mathsf{sk} = (z_{\mathsf{sk}} - z'_{\mathsf{sk}})/e$. From the second, $\alpha = (z_{\alpha} - z'_{\alpha})/e$. Substituting into the third equation yields $Z^{\alpha} \cdot \mathsf{req}^{\mathsf{sk}} = c_2$, so $(\mathsf{sk}, \alpha) \in \mathcal{R}_{\mathsf{Enc}}$.

Honest-verifier zero-knowledge. The simulator samples $c, z_{\mathsf{sk}}, z_{\alpha} \leftarrow\!\!\$\ \mathbb{Z}_p$ and sets $R_1 := g^{z_{\mathsf{sk}}} \mathsf{pk}^{-c}, R_2 := g^{z_{\alpha}} c_1^{-c}, R_3 := Z^{z_{\alpha}} \cdot \mathsf{req}^{z_{\mathsf{sk}}} \cdot c_2^{-c}$. The resulting transcript satisfies all verification equations and is identically distributed to an honest one, since the honest first-round values are uniformly random in $\mathbb{G}$ given the challenge.

Public inputs: a group $(\mathbb{G}, g, p)$ and $\mathbb{x} = (\mathsf{pk}, Z, \mathsf{req}, ct)$.
The prover provides a witness $\mathbb{w} = (\mathsf{sk}, \alpha)$ such that $\mathsf{pk} = g^{\mathsf{sk}}, \quad Z = g^{z}, \quad ct = (c_1, c_2) = (g^{\alpha},\ Z^{\alpha} \cdot \mathsf{req}^{\mathsf{sk}})$.
The verifier has no additional input.
Proving:

1. The prover samples $r_{\mathsf{sk}}, r_{\alpha} \leftarrow\!\!\$\ \mathbb{Z}_p$ and computes the commitments $R_1 := g^{r_{\mathsf{sk}}}, R_2 := g^{r_{\alpha}}, R_3 := Z^{r_{\alpha}} \cdot \mathsf{req}^{r_{\mathsf{sk}}}$. It sends (R_1, R_2, R_3) to the verifier.
2. The verifier samples a challenge $c \leftarrow\!\!\$\ \mathbb{Z}_p$ and sends it to the prover.
3. The prover computes the responses $z_{\mathsf{sk}} := r_{\mathsf{sk}} + c \cdot \mathsf{sk}, z_{\alpha} := r_{\alpha} + c \cdot \alpha$, and sends $(z_{\mathsf{sk}}, z_{\alpha})$ to the verifier.

Verification: Parse ct as (c_1, c_2). Upon receiving $(R_1, R_2, R_3, z_{\mathsf{sk}}, z_{\alpha})$, the verifier accepts iff

$$g^{z_{\mathsf{sk}}} \stackrel{?}{=} R_1 \cdot \mathsf{pk}^{c} \wedge g^{z_{\alpha}} \stackrel{?}{=} R_2 \cdot c_1^{c} \wedge Z^{z_{\alpha}} \cdot \mathsf{req}^{z_{\mathsf{sk}}} \stackrel{?}{=} R_3 \cdot c_2^{c}.$$

Fig. 14. Interactive Schnorr proof for the relation $\mathcal{R}_{\mathsf{Enc}}$.

Proofs of knowledge for commitment-opening, and secret keys. The relation $\mathcal{R}_{\mathsf{C}}$ is satisfied, if a party knows an opening (sk, ω) for a commitment $\mathsf{C} = g^{\mathsf{sk}} h^{\omega}$. The relation $\mathcal{R}_{\mathsf{DL}}$ is satisfied if a party knows the discrete logarithm sk of a group element pk.

$$\mathcal{R}_{\mathsf{C}} := \left\{ \begin{matrix} \mathbb{x} = (\mathsf{C}, \mathbb{G}, g, h) \\ \mathbb{w} = (\mathsf{sk}, \omega) \end{matrix} \ : \ \mathsf{C} = g^{\mathsf{sk}} h^{\omega} \right\}$$
$$\mathcal{R}_{\mathsf{DL}} := \left\{ \begin{matrix} \mathbb{x} = (\mathsf{pk}, \mathbb{G}, g) \\ \mathbb{w} = \mathsf{sk} \end{matrix} \ : \ \mathsf{pk} = g^{\mathsf{sk}} \right\}$$

We depict sigma protocols for these PoKs in Figures 15 and 16.

Public inputs: a group $(\mathbb{G}, g, p)$, an additional generator $h \in \mathbb{G}$, and a statement $\mathbb{x} = (\mathsf{C}, \mathbb{G}, g, h)$.
The prover provides a witness $\mathbb{w} = (\mathsf{sk}, \omega)$ such that $\mathsf{C} = g^{\mathsf{sk}} h^{\omega}$.
The verifier has no additional input.
Proving:

1. The prover samples $r_{\mathsf{sk}}, r_{\omega} \leftarrow\!\!\$\ \mathbb{Z}_p$ and computes the commitment $R := g^{r_{\mathsf{sk}}} h^{r_{\omega}}$. It sends R to the verifier.
2. The verifier samples a challenge $c \leftarrow\!\!\$\ \mathbb{Z}_p$ and sends it to the prover.
3. The prover computes the responses $z_{\mathsf{sk}} := r_{\mathsf{sk}} + c \cdot \mathsf{sk}, z_{\omega} := r_{\omega} + c \cdot \omega$, and sends $(z_{\mathsf{sk}}, z_{\omega})$ to the verifier.

Verification: Upon receiving $(R, z_{\mathsf{sk}}, z_{\omega})$, the verifier accepts iff $g^{z_{\mathsf{sk}}} h^{z_{\omega}} \stackrel{?}{=} R \cdot \mathsf{C}^{c}$.

Fig. 15. Interactive Schnorr proof for the relation $\mathcal{R}_{\mathsf{C}}$.

Lemma 2. *The cut-and-choose protocol in Figure 17 is complete, computationally sound under the DDH assumption, and honest-verifier zero-knowledge, assuming the HVZK of Lemma 3.*

Proof. Completeness. An honest prover with valid witness $(\mathsf{sk}, y_{\mathsf{tgt}})$ always passes all checks: for selected index j, the OR proof verifies that $T_j = \mathsf{H}_{\mathbb{G}}(y_{\mathsf{tgt}})^{\mathsf{sk} \cdot \alpha_j}$ is well-formed, the opening $c_j = \mathsf{H}_p(T_j) + r_j$ is consistent, and $R_j = g^{r_j}$. For unselected index k, the value $s_k = r_k + y_{\mathsf{win}}$ satisfies $g^{s_k} = Y_{\mathsf{win}} \cdot R_k$ and $A_k = g^{\alpha_k}$.

Soundness. Suppose no $y_{\mathsf{tgt}} \in [0, 2^{\ell}]$ exists such that y_{win} can be recovered evaluating the OPRF with key sk matching pk. Since all λ committed instances are constructed using the *same* y_{tgt}, no valid witness exists for

Public inputs: a group $(\mathbb{G}, g, p)$ and a statement $\mathbb{x} = (\mathsf{pk}, \mathbb{G}, g)$.
The prover provides a witness $\mathbb{w} = \mathsf{sk}$ such that $\mathsf{pk} = g^{\mathsf{sk}}$.
The verifier has no additional input.
Proving:

1. The prover samples $r \leftarrow\!\!\$\, \mathbb{Z}_p$ and computes $R := g^r$. It sends R to the verifier.
2. The verifier samples a challenge $c \leftarrow\!\!\$\, \mathbb{Z}_p$ and sends it to the prover.
3. The prover computes the response $z := r + c \cdot \mathsf{sk}$ and sends z to the verifier.

Verification: Upon receiving (R, z), the verifier accepts iff $g^z \stackrel{?}{=} R \cdot \mathsf{pk}^c$.

Fig. 16. Interactive Schnorr proof for the relation $\mathcal{R}_{\mathsf{DL}}$.

Public inputs: a group $(\mathbb{G}, g, p)$, and hash functions $\mathsf{H}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$, $\mathsf{H}_p : \{0,1\}^* \to \mathbb{Z}_p$, and a statement $\mathbb{x} = (\mathsf{pk}, Y_{\mathsf{win}}, \ell)$.
The prover provides $\mathbb{w} = (\mathsf{sk}, y_{\mathsf{tgt}})$ as additional input, and the verifier has no additional input.
Proving:

1. The prover samples scalars $(\alpha_1, \ldots, \alpha_\lambda, r_1, \ldots, r_\lambda) \leftarrow\!\!\$\, \mathbb{Z}_p$, computes the λ commitments
$$\left\{\mathsf{H}_p\big(\mathsf{H}_{\mathbb{G}}(y_{\mathsf{tgt}})^{\mathsf{sk}\cdot\alpha_i}\big) + r_i, R_i = g^{r_i}, A_i = g^{\alpha_i}\right\}_{1\le i\le\lambda},$$
and sends them to the verifier.
2. The verifier selects $\lambda/2$ random indices and sends them to the prover.
3. For each index j selected by the verifier, the prover outputs $(r_j, \mathsf{H}_{\mathbb{G}}(y)^{\mathsf{sk}\cdot\alpha_j})$, and a proof of well-formedness of $\mathsf{H}_{\mathbb{G}}(y)^{\mathsf{sk}\cdot\alpha_j}$ w.r.t. pk, g^{α_j}, and all possible values $\hat{y} \in \{0,1\}^\ell$ (depicted in Figure 18). In addition, for each unselected commitment with index k, the prover outputs $(\alpha_k, s_k = r_k + y_{\mathsf{win}})$.

Verification: The verifier obtains $r_j, \mathsf{H}_{\mathbb{G}}(y)^{\mathsf{sk}\cdot\alpha_j}$ and a proof of well-formedness for each selected index j, and the tuple (α_k, s_k) for each unselected index k.

1. For each selected index j, the verifier verifies the opening by checking the proof of well-formedness, and by asserting that r_j is a correct opening for R_j.
2. For each unselected index k, the verifier asserts that the value s_k is well-formed by checking if $Y_{\mathsf{win}} \cdot R_k \stackrel{?}{=} g^{s_k}$ and $A_k = g^{\alpha_k}$.

Fig. 17. Interactive cut-and-choose proof for $\mathcal{R}_{\mathsf{win}}$.

any of the selected instances' OR statements either. Hence for each selected index j the prover must either produce an accepting transcript of Lemma 3 without a valid witness, or guess the set of opened elements correctly. By the 2-special soundness of the sigma protocol, forging an OR proof succeeds with probability at most $1/p$. The probability of correctly guessing the opening is at most $\frac{((n/2)!)^2}{n!}$, which is also negligible. Soundness holds.

Honest-verifier zero-knowledge. The simulator draws the challenge set $\mathcal{I} \subset [\lambda]$ with $|\mathcal{I}| = \lambda/2$ uniformly and sets $y_{\mathsf{win}} := \log_g(Y_{\mathsf{win}})$.

– For each $j \in \mathcal{I}$: sample $r_j, \alpha_j \leftarrow\!\$\, \mathbb{Z}_p$, pick arbitrary $T_j \in \mathbb{G}$, set $c_j = \mathsf{H}_p(T_j) + r_j$, $R_j = g^{r_j}$, $A_j = g^{\alpha_j}$, and simulate the OR proof for T_j via Lemma 3.
– For each $k \notin \mathcal{I}$: sample $s_k, \alpha_k \leftarrow\!\$\, \mathbb{Z}_p$, set $R_k = g^{s_k}/Y_{\mathsf{win}}$, $A_k = g^{\alpha_k}$.

The opened instances provide no information about the target value y_{tgt} by the DDH assumption in $\mathbb{G}$. Furthermore, all commitments are uniformly distributed, and the revealed values are identically distributed in the honest execution.

Public inputs: a group $(\mathbb{G}, g, p)$, a hash function $\mathsf{H}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$, and a statement $\mathbb{x}_{\mathsf{or}} = (\mathsf{pk}, A, T, \ell)$.
The prover provides a witness $\mathbb{w}_{\mathsf{or}} = (\mathsf{sk}, \alpha, y)$ such that $y \in \{0,1\}^\ell$, $\mathsf{pk} = g^{\mathsf{sk}}$, $A = g^\alpha$, and $T = \mathsf{H}_{\mathbb{G}}(y)^{\mathsf{sk}\alpha}$.
The verifier has no additional input.
For every candidate $\hat{y} \in \{0,1\}^\ell$, define $h_{\hat{y}} := \mathsf{H}_{\mathbb{G}}(\hat{y})$. The protocol proves that

$$\exists \hat{y} \in \{0,1\}^\ell,\ \exists (\mathsf{sk}, \alpha) \in \mathbb{Z}_p^2 : \mathsf{pk} = g^{\mathsf{sk}},\ A = g^\alpha,\ T = h_{\hat{y}}^{\mathsf{sk}\alpha}.$$

Proving:

1. For the real branch $\hat{y} = y$, the prover computes $B_y := h_y^\alpha$. It samples $t_\alpha, t_{\mathsf{sk}} \leftarrow\!\$\, \mathbb{Z}_p$ and computes
$$a_{1,y} = g^{t_\alpha}, \qquad a_{2,y} = h_y^{t_\alpha}, \qquad a_{3,y} = g^{t_{\mathsf{sk}}}, \qquad a_{4,y} = B_y^{t_{\mathsf{sk}}}.$$
2. For every simulated branch $\hat{y} \in \{0,1\}^\ell \setminus \{y\}$, the prover samples $c_{\hat{y}}, z_{\alpha,\hat{y}}, z_{\mathsf{sk},\hat{y}}, \beta_{\hat{y}} \leftarrow\!\$\, \mathbb{Z}_p$, defines $B_{\hat{y}} := h_{\hat{y}}^{\beta_{\hat{y}}}$, and computes
$$a_{1,\hat{y}} = g^{z_{\alpha,\hat{y}}} A^{-c_{\hat{y}}}, \qquad a_{2,\hat{y}} = h_{\hat{y}}^{z_{\alpha,\hat{y}}} B_{\hat{y}}^{-c_{\hat{y}}}, \qquad a_{3,\hat{y}} = g^{z_{\mathsf{sk},\hat{y}}}\, \mathsf{pk}^{-c_{\hat{y}}}, \qquad a_{4,\hat{y}} = B_{\hat{y}}^{z_{\mathsf{sk},\hat{y}}} T^{-c_{\hat{y}}}.$$
3. The prover sends, for every $\hat{y} \in \{0,1\}^\ell$, the first message $\left(B_{\hat{y}}, a_{1,\hat{y}}, a_{2,\hat{y}}, a_{3,\hat{y}}, a_{4,\hat{y}}\right)$ to the verifier.
4. The verifier samples a challenge $c \leftarrow\!\$\, \mathbb{Z}_p$ and sends it to the prover.
5. The prover sets $c_y := c - \sum_{\hat{y} \in \{0,1\}^\ell \setminus \{y\}} c_{\hat{y}}$, and computes the real responses
$$z_{\alpha,y} := t_\alpha + c_y \alpha, \qquad z_{\mathsf{sk},y} := t_{\mathsf{sk}} + c_y \mathsf{sk}.$$
6. The prover sends, for every $\hat{y} \in \{0,1\}^\ell$, the tuple $(c_{\hat{y}}, z_{\alpha,\hat{y}}, z_{\mathsf{sk},\hat{y}})$ to the verifier.

Verification: Upon receiving $\left(B_{\hat{y}}, a_{1,\hat{y}}, a_{2,\hat{y}}, a_{3,\hat{y}}, a_{4,\hat{y}}, c_{\hat{y}}, z_{\alpha,\hat{y}}, z_{\mathsf{sk},\hat{y}}\right)_{\hat{y} \in \{0,1\}^\ell}$, the verifier accepts iff:

1. The branch challenges sum to the global challenge: $\sum_{\hat{y} \in \{0,1\}^\ell} c_{\hat{y}} \equiv c \pmod p$.
2. For every $\hat{y} \in \{0,1\}^\ell$, the following four equations hold:
$$g^{z_{\alpha,\hat{y}}} \stackrel{?}{=} a_{1,\hat{y}} \cdot A^{c_{\hat{y}}}, \qquad h_{\hat{y}}^{z_{\alpha,\hat{y}}} \stackrel{?}{=} a_{2,\hat{y}} \cdot B_{\hat{y}}^{c_{\hat{y}}}, \qquad g^{z_{\mathsf{sk},\hat{y}}} \stackrel{?}{=} a_{3,\hat{y}} \cdot \mathsf{pk}^{c_{\hat{y}}}, \qquad B_{\hat{y}}^{z_{\mathsf{sk},\hat{y}}} \stackrel{?}{=} a_{4,\hat{y}} \cdot T^{c_{\hat{y}}}.$$

Fig. 18. Interactive Schnorr OR proof for well-formedness of cut-and-choose openings (Figure 17).

Lemma 3. *The sigma protocol in Figure 18 is complete, 2-special sound with knowledge error $1/p$, and honest-verifier zero-knowledge.*

Proof. The protocol follows the standard Cramer–Damgård–Schoenmakers OR-composition.

Completeness. For the real branch $\hat{y} = y$, the four verification equations hold by expanding $a_{1,y}, \ldots, a_{4,y}$ and the responses $z_{\alpha,y}, z_{\mathsf{sk},y}$. For every simulated branch $\hat{y} \neq y$, the equations hold by construction.

Special soundness. Given two accepting transcripts with identical first messages $\{(B_{\hat{y}}, a_{1,\hat{y}}, \ldots, a_{4,\hat{y}})\}_{\hat{y}}$ but distinct global challenges $c \neq c'$, there exists $y^* \in \{0,1\}^\ell$ with $c_{y^*} \neq c'_{y^*}$, since $\sum_{\hat{y}} c_{\hat{y}} = c$ and $\sum_{\hat{y}} c'_{\hat{y}} = c'$. Let $e = c_{y^*} - c'_{y^*} \neq 0$. From the first two equations for branch y^*:

$$\alpha = \frac{z_{\alpha,y^*} - z'_{\alpha,y^*}}{e}, \qquad \text{satisfying} \quad A = g^\alpha \ \text{ and } \ B_{y^*} = h_{y^*}^\alpha.$$

From the last two:

$$\mathsf{sk} = \frac{z_{\mathsf{sk},y^*} - z'_{\mathsf{sk},y^*}}{e}, \qquad \text{satisfying} \quad \mathsf{pk} = g^{\mathsf{sk}} \ \text{ and } \ T = B_{y^*}^{\mathsf{sk}} = h_{y^*}^{\mathsf{sk}\alpha}.$$

Hence $(\mathsf{sk}, \alpha, y^*)$ is a valid witness.

Honest-verifier zero-knowledge. All simulated branches use the standard Schnorr simulation (responses and branch challenges uniform). The real branch contributes a pair of composed Chaum–Pedersen transcripts, each perfectly simulatable by choosing responses first. The joint distribution is therefore identical to a simulated transcript.

Lemma 4. *The sigma protocol in Figure 19 is complete, 2-special sound with knowledge error $1/p$, and honest-verifier zero-knowledge.*

Proof. The argument is structurally identical to Lemma 3. Each branch proves knowledge of (sk, ρ) such that $\mathsf{pk} = g^{\mathsf{sk}}$, $M_{\hat{y}} = h_{\hat{y}}^{\mathsf{sk}}$, $U = u^\rho$, and $V/M_{\hat{y}} = g^\rho$ via a pair of composed Chaum–Pedersen sub-arguments sharing the branch challenge $c_{\hat{y}}$.

Special soundness. From two transcripts with distinct global challenges, there exists $y^* \in \{0,1\}^\ell$ with $c_{y^*} \neq c'_{y^*}$. From the first two equations, $\mathsf{sk} = (z_{\mathsf{sk},y^*} - z'_{\mathsf{sk},y^*})/(c_{y^*} - c'_{y^*})$, satisfying $\mathsf{pk} = g^{\mathsf{sk}}$ and $M_{y^*} = h_{y^*}^{\mathsf{sk}}$. From the last two, $\rho = (z_{\rho,y^*} - z'_{\rho,y^*})/(c_{y^*} - c'_{y^*})$, satisfying $U = u^\rho$ and $V = M_{y^*} \cdot g^\rho = h_{y^*}^{\mathsf{sk}} \cdot g^\rho$.

Honest-verifier zero-knowledge. Follows by the same OR-simulation argument as Lemma 3.

Lemma 5. *The sigma protocol in Figure 20 is complete, 2-special sound with knowledge error $1/p$, and honest-verifier zero-knowledge.*

Proof. The protocol is a conjunction of three Chaum–Pedersen sub-arguments sharing a single challenge c.

Completeness. For a valid witness (h, ρ, α), set $\delta := \rho\alpha$. Then $T_g = X^\alpha / Y = (hg^\rho)^\alpha / h^\alpha = g^{\rho\alpha} = g^\delta$. The responses $z_1 = r_1 + c\alpha$, $z_2 = r_2 + c\delta$, $z_3 = r_3 + c\alpha$ satisfy all six verification equations by direct computation.

Special soundness. Given two accepting transcripts with the same first message but distinct challenges $c \neq c'$, let $e = c - c'$. From the first sub-argument, $\alpha = (z_1 - z'_1)/e$. From the second, $\delta = (z_2 - z'_2)/e$, hence $\rho = \delta/\alpha$. Setting $h = X \cdot g^{-\rho}$, the third sub-argument confirms $h^\alpha = Y$, so (h, ρ, α) is a valid witness.

Honest-verifier zero-knowledge. The simulator samples $c, z_1, z_2, z_3 \leftarrow\!\!\$\ \mathbb{Z}_p$, sets $T_g \leftarrow\!\!\$\ \mathbb{G}$, and computes

$$\begin{aligned} a_{1,1} &:= u^{z_1} U_\alpha^{-c}, & a_{1,2} &:= U_\rho^{z_1} T_u^{-c}, \\ a_{2,1} &:= u^{z_2} T_u^{-c}, & a_{2,2} &:= g^{z_2} T_g^{-c}, \\ a_{3,1} &:= u^{z_3} U_\alpha^{-c}, & a_{3,2} &:= X^{z_3} (YT_g)^{-c}. \end{aligned}$$

Each pair is a perfect Chaum–Pedersen simulation. Since the commitment pairs are uniformly distributed in $\mathbb{G}$, the simulated transcript is identically distributed to an honest one.

Public inputs: a group $(\mathbb{G}, g, p)$, a second generator $u \in \mathbb{G}$, a hash function $\mathsf{H}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$, and a statement $\mathbb{x}_{\mathsf{or}} = (\mathsf{pk}, U, V, \ell)$. For every candidate $\hat{y} \in \{0,1\}^\ell$, define $h_{\hat{y}} := \mathsf{H}_{\mathbb{G}}(\hat{y})$.
The prover provides a witness $\mathbb{w}_{\mathsf{or}} = (\mathsf{sk}, \rho, y)$ such that $y \in \{0,1\}^\ell$, $\mathsf{pk} = g^{\mathsf{sk}}$, and

$$U = u^\rho, \qquad V = h_y^{\mathsf{sk}} \cdot g^\rho.$$

The protocol proves that $\exists \hat{y} \in \{0,1\}^\ell,\ \exists (\mathsf{sk}, \rho) \in \mathbb{Z}_p^2 : \mathsf{pk} = g^{\mathsf{sk}} \ \wedge\ U = u^\rho \ \wedge\ V = h_{\hat{y}}^{\mathsf{sk}} \cdot g^\rho$.
Proving:

1. For the real branch $\hat{y} = y$, the prover computes $M_y := h_y^{\mathsf{sk}}$. It samples $t_{\mathsf{sk}}, t_\rho \leftarrow\!\$\ \mathbb{Z}_p$ and computes

$$a_{1,y} = g^{t_{\mathsf{sk}}}, \qquad a_{2,y} = h_y^{t_{\mathsf{sk}}}, \qquad a_{3,y} = u^{t_\rho}, \qquad a_{4,y} = g^{t_\rho}.$$

2. For every simulated branch $\hat{y} \in \{0,1\}^\ell \setminus \{y\}$, the prover samples $c_{\hat{y}}, z_{\mathsf{sk},\hat{y}}, z_{\rho,\hat{y}}, \mu_{\hat{y}} \leftarrow\!\$\ \mathbb{Z}_p$, defines $M_{\hat{y}} := h_{\hat{y}}^{\mu_{\hat{y}}}$, and computes

$$a_{1,\hat{y}} = g^{z_{\mathsf{sk},\hat{y}}}\, \mathsf{pk}^{-c_{\hat{y}}}, \qquad a_{2,\hat{y}} = h_{\hat{y}}^{z_{\mathsf{sk},\hat{y}}}\, M_{\hat{y}}^{-c_{\hat{y}}}, \qquad a_{3,\hat{y}} = u^{z_{\rho,\hat{y}}} U^{-c_{\hat{y}}}, \qquad a_{4,\hat{y}} = g^{z_{\rho,\hat{y}}} (V/M_{\hat{y}})^{-c_{\hat{y}}}.$$

3. The prover sends, for every $\hat{y} \in \{0,1\}^\ell$, the first message $\left(M_{\hat{y}}, a_{1,\hat{y}}, a_{2,\hat{y}}, a_{3,\hat{y}}, a_{4,\hat{y}}\right)$ to the verifier.
4. The verifier samples a challenge $c \leftarrow\!\$\ \mathbb{Z}_p$ and sends it to the prover.
5. The prover sets $c_y := c - \sum_{\hat{y} \in \{0,1\}^\ell \setminus \{y\}} c_{\hat{y}} \pmod p$ and computes

$$z_{\mathsf{sk},y} := t_{\mathsf{sk}} + c_y \mathsf{sk}, \qquad z_{\rho,y} := t_\rho + c_y \rho.$$

6. The prover sends, for every $\hat{y} \in \{0,1\}^\ell$, the tuple $(c_{\hat{y}}, z_{\mathsf{sk},\hat{y}}, z_{\rho,\hat{y}})$ to the verifier.

Verification: Upon receiving $\left(M_{\hat{y}}, a_{1,\hat{y}}, a_{2,\hat{y}}, a_{3,\hat{y}}, a_{4,\hat{y}}, c_{\hat{y}}, z_{\mathsf{sk},\hat{y}}, z_{\rho,\hat{y}}\right)_{\hat{y} \in \{0,1\}^\ell}$, the verifier accepts iff:

1. $\sum_{\hat{y} \in \{0,1\}^\ell} c_{\hat{y}} \equiv c \pmod p$.
2. For every $\hat{y} \in \{0,1\}^\ell$, the following equations hold:

$$g^{z_{\mathsf{sk},\hat{y}}} \stackrel{?}{=} a_{1,\hat{y}} \cdot \mathsf{pk}^{c_{\hat{y}}}, \qquad h_{\hat{y}}^{z_{\mathsf{sk},\hat{y}}} \stackrel{?}{=} a_{2,\hat{y}} \cdot M_{\hat{y}}^{c_{\hat{y}}}, \qquad u^{z_{\rho,\hat{y}}} \stackrel{?}{=} a_{3,\hat{y}} \cdot U^{c_{\hat{y}}}, \qquad g^{z_{\rho,\hat{y}}} \stackrel{?}{=} a_{4,\hat{y}} \cdot (V/M_{\hat{y}})^{c_{\hat{y}}}.$$

Fig. 19. Interactive Schnorr OR proof that (U, V) is well formed as $U = u^\rho$ and $V = h_y^{\mathsf{sk}} \cdot g^\rho$ for some $y \in \{0,1\}^\ell$.

Public inputs: a cyclic group $(\mathbb{G}, g, p)$ of prime order p, a second generator $u \in \mathbb{G}$, and a statement $\mathbb{x}_{\mathsf{wf}} = (X, Y, U_\rho, U_\alpha, T_u, T_g)$. The prover provides a witness $\mathbb{w}_{\mathsf{wf}} = (h, \rho, \alpha)$ such that $X = h \cdot g^\rho$, $Y = h^\alpha$, $U_\rho = u^\rho$, $U_\alpha = u^\alpha$, and $T_u = u^{\rho\alpha}$. The verifier has no additional input.
The protocol proves that

$$\exists (h, \rho, \alpha) \in \mathbb{G} \times \mathbb{Z}_p^2 : X = h \cdot g^\rho,\ Y = h^\alpha,\ U_\rho = u^\rho,\ U_\alpha = u^\alpha,\ T_u = u^{\rho\alpha}.$$

The prover first defines the auxiliary value $T_g := X^\alpha / Y$. For a valid witness, this satisfies $T_g = g^{\rho\alpha}$. The proof consists of three Chaum–Pedersen subproofs:

$$\log_u(U_\alpha) = \log_{U_\rho}(T_u), \qquad \log_u(T_u) = \log_g(T_g), \qquad \log_u(U_\alpha) = \log_X(YT_g).$$

Proving:

1. The prover computes $T_g := X^\alpha / Y$ and $\delta := \rho\alpha \in \mathbb{Z}_p$.
2. For the first subproof $\log_u(U_\alpha) = \log_{U_\rho}(T_u)$, the prover samples $r_1 \leftarrow\!\$\, \mathbb{Z}_p$ and computes

$$a_{1,1} := u^{r_1}, \qquad a_{1,2} := U_\rho^{r_1}.$$

3. For the second subproof $\log_u(T_u) = \log_g(T_g)$, the prover samples $r_2 \leftarrow\!\$\, \mathbb{Z}_p$ and computes

$$a_{2,1} := u^{r_2}, \qquad a_{2,2} := g^{r_2}.$$

4. For the third subproof $\log_u(U_\alpha) = \log_X(YT_g)$, the prover samples $r_3 \leftarrow\!\$\, \mathbb{Z}_p$ and computes

$$a_{3,1} := u^{r_3}, \qquad a_{3,2} := X^{r_3}.$$

5. The prover sends $(T_g, a_{1,1}, a_{1,2}, a_{2,1}, a_{2,2}, a_{3,1}, a_{3,2})$ to the verifier.
6. The verifier samples a challenge $c \leftarrow\!\$\, \mathbb{Z}_p$ and sends it to the prover.
7. The prover computes

$$z_1 := r_1 + c\alpha, \qquad z_2 := r_2 + c\delta, \qquad z_3 := r_3 + c\alpha$$

and sends (z_1, z_2, z_3) to the verifier.

Verification: Upon receiving $(T_g, a_{1,1}, a_{1,2}, a_{2,1}, a_{2,2}, a_{3,1}, a_{3,2}, z_1, z_2, z_3)$, the verifier accepts iff all six equations hold:

$$u^{z_1} \stackrel{?}{=} a_{1,1} \cdot U_\alpha^c, \qquad U_\rho^{z_1} \stackrel{?}{=} a_{1,2} \cdot T_u^c, \qquad u^{z_2} \stackrel{?}{=} a_{2,1} \cdot T_u^c, \qquad g^{z_2} \stackrel{?}{=} a_{2,2} \cdot T_g^c, \qquad u^{z_3} \stackrel{?}{=} a_{3,1} \cdot U_\alpha^c,\ X^{z_3} \stackrel{?}{=} a_{3,2} \cdot (YT_g)^c.$$

If all checks pass, then the verifier concludes that $T_u = u^{\rho\alpha}$, $T_g = g^{\rho\alpha}$, and $YT_g = X^\alpha$; hence

$$Y \cdot g^{\rho\alpha} = X^\alpha = (hg^\rho)^\alpha = h^\alpha g^{\rho\alpha},$$

which implies $Y = h^\alpha$ while h remains hidden.

Fig. 20. Interactive Chaum–Pedersen proof for well-formedness of (X, Y) with hidden base h.