Threshold Hardened Derivation for BIP32-Style Wallets from the Hydra PRF

Cemre Özdoğan* Oğuz Yayla†

22 November 2026

Abstract

BIP32 hardened key derivation computes HMAC-SHA512 on the parent private key and chain code. In a threshold wallet these are secret-shared, so the hash must be evaluated inside a multi-party computation. We present a hardened derivation that replaces HMAC-SHA512 by the arithmetization-oriented PRF Hydra. Once the PRF output is serialized, every remaining step is BIP32's; the derived keys still differ from BIP32's, so the scheme is a sibling of BIP32 and not wire-compatible with it. Both parent secrets form the PRF key, only public data enters its input, and every hardened child stays secret-shared and can derive further. We prove parent–child hiding, sibling independence, and hardened-child unlinkability by one path hybrid. Because the parent key is both part of the PRF key and BIP32's additive offset, the proofs need an offset-PRF assumption, which the published PRF claim of Hydra does not imply. The threshold protocol is proved UC-secure in the arithmetic-black-box model against static semi-honest adversaries, and with abort against static malicious ones, with no assumption on the PRF. With five parties under MP-SPDZ's malicious Shamir protocol, which our proofs do not cover, one derivation needs 243 rounds and 0.33 s, against 15,226 rounds and 3.3 s for HMAC-SHA512; at 10 ms round-trip time, this becomes 1.9 s against 91 s.

Keywords: MPC · Threshold Wallets · BIP32 · Arithmetization-Oriented PRF · Hydra · Megafono

1 Introduction

Threshold wallets split a signing key among several devices, so that an attacker must corrupt several of them to steal funds. Most wallets also follow BIP32 [29], which derives a whole tree of keys from one root. Its *hardened* derivation, the mode that keeps a leaked child from exposing its parent, computes HMAC-SHA512 on the parent's private key and chain code. In a threshold wallet those are secret-shared, so hardened derivation requires SHA-512 inside a multi-party computation (MPC). That is expensive. In our measurements, with five parties and security against malicious parties, one such derivation needs 15,226 communication rounds and takes 3.3 s on one machine, and 91 s at a network round-trip time of 10 ms (§7).

The starting point is that BIP32 uses HMAC-SHA512 as a pseudorandom function (PRF), not as a MAC. We therefore replace it by a PRF designed for MPC: Hydra [20], which works over a prime field $\mathbb{F}_p$, so that almost all of its operations are local on secret shares. Once the PRF's four output words are serialized, which adds one more reason to reject an index (§4.3), everything stays BIP32's own: the split of the output into $I_L \parallel I_R$, the child key $(I_L + k_{par}) \bmod n$, the chain code I_R, and the rule that a rejected index is skipped. Unlike BIP32, both parent secrets go into the PRF's key and only public data into its input (§4.1).

*Middle East Technical University, cemre.ozdogan@metu.edu.tr
†Middle East Technical University, oguz@metu.edu.tr

The closest prior work is that of Das et al. [13], who replace the hash by a DDH-based threshold verifiable random function. Their derivation is non-interactive and far cheaper than ours, but their hardened nodes are single devices and cannot derive further. In our construction every hardened child stays secret-shared, so a hardened node can itself derive and the whole tree is threshold, as in BIP32. Section 3 compares the two in detail.

We make four contributions.

- *Construction* (§4): hardened derivation with Hydra, including the fixed-width serialization of its output and BIP32's addition modulo the secp256k1 order, all computed on secret shares.
- *Function-level security* (§5): parent–child hiding, sibling independence and hardened-child unlinkability, proved by a single path hybrid over a GGM-style tree under an explicitly stated offset-PRF assumption, together with the exact part of it that the published Hydra PRF claim covers. The hybrid reduces the security of the tree to a property of a single node: its derivations look random even to someone who holds the node's public key. For Hydra this property is an assumption beyond the published PRF claim.
- *Protocol-level security* (§6): the threshold protocol UC-realizes an ideal derivation functionality against static semi-honest adversaries, and with abort against static malicious adversaries, leaking only the public rejection verdicts, with no assumption on the PRF. The malicious result carries over to any protocol that realizes the black box with abort; we do not show that MP-SPDZ's malicious Shamir protocol, which we benchmark, is one.
- *Implementation and evaluation* (§7): an MP-SPDZ implementation, checked against an independent reference implementation, and a comparison with HMAC-SHA512 in the same setting as Das et al. [13]: five parties under malicious Shamir sharing. One derivation needs 243 rounds (by the round counter of the measured implementation) and 0.33 s; at 10 ms round-trip time it takes 1.9 s, against the HMAC-SHA512 figures above. We also report semi-honest Shamir sharing and MASCOT.

The work has clear limits. The derivation is not wire-compatible with BIP32: hardened children differ from those a standard wallet derives. Security against malicious parties is security with abort, and it is proved for any protocol that realizes the black box with abort (Corollary 6.13). MP-SPDZ's malicious Shamir protocol, which we benchmark, follows the method of Lindell and Nof [26] for multiplication and opening but produces random bits differently, and we do not show that it is such a protocol (Remark 6.15). The protocol results hold for callers that give all parties the same commands, one at a time (§6.1), and they model root generation and public-key computation as ideal commands. The function-level proofs rest on the offset-PRF assumption (Assumption 5.21), which is new and is not implied by the published security claim of Hydra. A deployment that publishes a chain code needs a further assumption (Assumption 5.27). Our benchmarks ran on one machine with an emulated network. Under semi-honest Shamir sharing, with MP-SPDZ's default batch size for preprocessing, the construction sends about 7× more data than the HMAC-SHA512 baseline (under malicious Shamir sharing, 2.4× less), and under MASCOT without network delay HMAC-SHA512 is faster.

Section 2 recalls BIP32 hardened derivation and Hydra, and Section 3 discusses related work. Section 4 gives the construction, Section 5 the security of the derivation function, Section 6 the threshold protocol and its security against semi-honest and malicious adversaries, and Section 7 the implementation and the evaluation. Section 8 concludes. The appendices contain the full Hydra specification, the compile listing, the child-scalar procedure, the full UC proof and additional measurements.

2 Preliminaries

2.1 BIP32 hardened derivation

For a parent private key k_{par}, parent chain code c_{par}, and a hardened index $i \in [2^{31}, 2^{32})$, BIP32 computes

$$I \;=\; \text{HMAC-SHA512}\big(\underbrace{c_{par}}_{\text{key}},\; \underbrace{0x00 \,\|\, \text{ser}_{256}(k_{par}) \,\|\, \text{ser}_{32}(i)}_{\text{data}}\big),$$

splits $I = I_L \,\|\, I_R$ into two 256-bit halves, and sets the child key and chain code to

$$k_{child} \;=\; (I_L + k_{par}) \bmod n, \qquad c_{child} \;=\; I_R,$$

where n is the secp256k1 group order. If $I_L \geq n$ or $k_{child} = 0$, the resulting key is invalid, and BIP32 proceeds with the next index. In BIP32 the chain code *keys* the hash and the private key sits in the *data*. We do not preserve that internal assignment; §4.1 explains why, and §4.2 fixes the one we use instead. The *interface* (the inputs consumed, the $I = I_L \| I_R$ split, and the child key and chain-code roles) is preserved exactly.

2.2 The Hydra PRF

Hydra [20] is a pseudorandom function over a prime field $\mathbb{F}_p$ that follows the Megafono design. We use its design and round numbers as specified by its authors. The instance is ours: the prime, and with it the matrices M_I and M_J, which we chose to meet the conditions of [20, §5.2, §5.4] rather than generate with the procedure of the reference code. Appendix A gives the full specification and our instance; this subsection recalls only what the rest of the paper needs.

Hydra is keyed by $K \in \mathbb{F}_p^4$ and maps an input $m = [\nu \,\|\, IV] \in \mathbb{F}_p^4$, consisting of a nonce $\nu \in \mathbb{F}_p$ and a fixed initial value $IV \in \mathbb{F}_p^3$, to a sequence of output words in $\mathbb{F}_p$. We use four output words and write $E_K(m) \in \mathbb{F}_p^4$ for them. Its security as a PRF is claimed for an attacker whose data is limited to $2^{\kappa/2}$ outputs per key [20, §5.1].

An evaluation has three parts. The input is first whitened with the key, $s \leftarrow M_E \cdot (m + K)$. A keyed permutation of $\mathbb{F}_p^4$, the *body*, is then applied: 2 external rounds, R_I internal rounds and 4 further external rounds, so $R_E = 6$ external rounds in total. Besides its final state y, to which K is added, the body returns the sum z of its intermediate states, which extends the state to $\mathbb{F}_p^8$ at no multiplicative cost. A second keyed permutation, the *head*, is then applied to $v = y \,\|\, z$. It has R_H rounds, adds the expanded key $K' = K \,\|\, (M_E \cdot K) \in \mathbb{F}_p^8$ after every round, and its output is added to v (feed-forward). The first four words of the result are $E_K(m)$.

Each round is a nonlinear layer followed by a fixed matrix and a round constant, and the nonlinear layers are the only non-linear operations of the design. An external round cubes each of the four state words (S_E), an internal round performs two sequential squarings (S_I), and a head round performs one squaring (S_J). The key schedule is linear.

We instantiate Hydra at the security level $\kappa = 128$ over the prime $p = 2^{128} + 385$, for which the round numbers of [20] are $R_I = 42$ and $R_H = 39$. The prime is chosen so that every 128-bit word embeds into $\mathbb{F}_p$ without reduction and cubing is a permutation of $\mathbb{F}_p$ (§A.1). The matrices are given in §A.2.

2.3 MPC cost model

We use a linear secret-sharing scheme over the same field $\mathbb{F}_p$, additive or Shamir [27], and write $[\![x]\!]$ for a sharing of x and $[\![x]\!]_j$ for the share held by party P_j. Linear combinations of shared values with public coefficients are computed locally, without communication. A multiplication of two shared values costs one round of communication; in SPDZ-type protocols and in malicious Shamir sharing it consumes one preprocessed Beaver triple [6]. Multiplications that do not

depend on each other share a round. Communication is therefore the cost, and it is driven by the number and the depth of the multiplications.

In Hydra, every matrix multiplication, round-constant addition, key addition and key injection, the sum z and the key expansion are linear, hence local. Only the three nonlinear layers communicate: S_E costs 4 multiplications per round, S_I costs 2 and S_J costs 1. At our parameters one evaluation of E_K takes $4R_E + 2R_I + R_H = 147$ multiplications. These are Beaver triples; the 24 cubes also use one square pair each, and the Hydra paper counts the two kinds together as the 171 preprocessed elements of this instance [20, Table 1]. The multiplicative depth is 130: one round per external round and per head round and two per internal round, that is $R_E + 2R_I + R_H = 129$, and one more round for the 24 data-independent products that the square-pair method for cubing needs (Appendix B). The Hydra paper lists depth 131 for the same round numbers [20, Table 1]: its benchmark program adds one round for opening the output, which our derivation never does, since its output stays shared. Because key, state and sharing all live in $\mathbb{F}_p$, no conversion between arithmetic and Boolean sharings is needed inside the PRF.

2.4 Parameters and notation

The following parameters and symbols are fixed throughout.

Symbol	Definition
$p = 2^{128} + 385$	Prime modulus (129-bit). See §A.1.
S-box exponent 3	Smallest odd integer ≥ 3 that is coprime to $p - 1$.
$K \in \mathbb{F}_p^4$	PRF key, secret-shared as $[\![K]\!]$.
$R_E = 6$	Total external rounds: 2 initial + 4 final (Hades asymmetric split).
$R_I = 42$	Internal rounds for $\kappa = 128$ at the prime of §A.1, per [20].
$R_H = 39$	Head rounds for $\kappa = 128$ at the same prime, per [20].
$M_E \in \mathbb{F}_p^{4\times 4}$	MDS matrix for external rounds. See §A.2.
$M_I \in \mathbb{F}_p^{4\times 4}$	Matrix for internal rounds; satisfies conditions (a)–(c) of [20, §5.2].
$M_J \in \mathbb{F}_p^{8\times 8}$	Matrix for head rounds; satisfies analogous conditions [20, §5.4].
$K' \in \mathbb{F}_p^8$	Expanded key: $K' = K \,\|\, (M_E \cdot K)$.
n	Order of the secp256k1 group [9] (256-bit). See §4.4.
$\varepsilon_{\text{ovf}} = 1 - (2^{128}/p)^4$	Overflow-band probability, $\leq 4 \cdot 385/p \approx 2^{-117.4}$. See §4.3.
$\varepsilon_{\text{ord}} = 1 - n/2^{256}$	Out-of-range probability, $\approx 2^{-127.7}$. See §4.4.
$\varepsilon_{\text{stat}} = \varepsilon_{\text{ovf}} + \varepsilon_{\text{ord}}$	Combined statistical defect, $\approx 2^{-117.4} < 2^{-115}$. See §4.3.
N, T	Number of parties and corruption threshold.
$[\![x]\!]$, $[\![x]\!]_j$	Sharing of x; share held by party P_j.
k, c	Private key and chain code of a node.
$\omega_0, \ldots, \omega_3$	Output words of E_K. See §4.3.

3 Related work

Das, Erwig, Faust, Lehwalder, Loss, Qu and Riahi [13] address the same problem: BIP32 hardened derivation needs a hash of the shared secret key, which is expensive to compute in MPC. They replace the hash by the DDH-based threshold verifiable random function of Galindo et al. [17], which re-uses the threshold ECDSA key shares of the wallet. Their derivation is non-interactive, and its security is proved in the random-oracle model. They let the adversary corrupt

a minority of the devices of a non-hardened node, and argue that hardened derivation must be secure against malicious adversaries. Their non-hardened derivation follows BIP32 exactly; their hardened derivation, like ours, does not reproduce BIP32's HMAC output. Their hardened nodes are single devices, not thresholdized, and cannot derive further: hardened wallets "always represent leaves" of the tree [13, §5]. They compare with a SHA-512-based derivation in MP-SPDZ under malicious Shamir sharing, on one machine with a delay of 10 ms per communication round [13, §6]. With threshold 2 of 5 devices their derivation takes 22.12 ms, against 68,694.51 ms for the MPC baseline (1,404.51 ms without the delay), and the parties exchange 0.64 KB in total, against 104.59 MB. The MPC baseline needs more than 6000 communication rounds.

This work differs from theirs in several respects. Our hardened children stay secret-shared, so a hardened node can itself derive, and the whole tree is threshold, as in BIP32; in [13] the tree ends at a hardened node held by one device. Our security rests on symmetric-key assumptions about Hydra (Assumption 5.21), theirs on DDH in the random-oracle model. Both target malicious adversaries with an honest majority; ours is proved secure with abort for any protocol that realizes a black box with abort; we do not show that a published protocol does, and the MP-SPDZ implementation we benchmark is not shown to be one (Remark 6.15). We benchmark it under the same protocol as their SHA-512 baseline (§7). Their derivation is far cheaper than ours: with a delay of 10 ms per communication round it takes 22 ms and 0.64 KB in total, whereas ours takes 1.9 s at a 10 ms round-trip time and 108.6 MB in total (§7). We pay this cost to keep the hardened child secret-shared. The two SHA-512 baselines, both under malicious Shamir sharing, theirs at about 69 s and ours at 91 s, are of the same order of magnitude; without delay, theirs takes 1,404.51 ms and 104.59 MB in total, ours 3.32 s and 267 MB. All of these comparisons, the first included, mix different hardware (14 cores in [13], a 4-core laptop here) and different ways of emulating delay (a delay per communication round in [13], a round-trip time on the network interface here). The two baselines also compute different things: their SHA-512-based derivation, and our full HMAC-SHA512 with four compressions and the addition modulo n. The numbers show orders of magnitude and should not be compared more closely.

Das, Erwig, Faust, Loss and Riahi [12] were the first to formally model and analyze BIP32 wallets. Chuang, Hsu and Lee [10] translate BIP32 key generation and hardened derivation to the two-party setting. Groth and Shoup [21] give a threshold ECDSA scheme with BIP32's additive key rerandomization, i.e. non-hardened derivation, and do not consider hardened derivation.

A line of symmetric primitives is designed to be cheap in MPC over $\mathbb{F}_p$: MiMC [1] minimizes the number of multiplications; GMiMC [2] builds Feistel structures from it; the HADES strategy [19], instantiated as HadesMiMC, mixes full and partial S-box rounds; Rescue [3] alternates a power map with its inverse; Ciminion [14] uses Toffoli gates in a Farfalle-like mode; and Hydra [20] follows the Megafono design. They trade the number of multiplications against depth differently. Under network latency the depth matters most, and the Hydra designers note that Hydra has a larger depth than HadesMiMC and Rescue [20, §8]. This work does not compare the primitives; it uses Hydra and measures it against HMAC-SHA512, the function it replaces.

4 Construction

BIP32 never verifies a tag, so HMAC-SHA512 serves there only *as a PRF*, and we replace it directly with the Hydra PRF E_K of §2.2, keeping BIP32's inputs, its $I = I_L \| I_R$ split, and its child-key and chain-code roles unchanged, while re-assigning which secret occupies the PRF's key slot (§4.1). The result is a *sibling* to BIP32: it has the same roles and interface, but different bytes. Staying wire-exact would require SHA-512 inside MPC, which is the cost we set out to avoid.

4.1 The key and input assignment

BIP32's placement of c_{par} in the key slot and k_{par} in the data slot is not a requirement of hardened derivation; it is a consequence of HMAC's interface. HMAC offers one key slot and one variable-length message slot, and of the two parent secrets the chain code is the one that naturally fits the former. HYDRA has a different interface: its key space $\mathbb{F}_p^4$ carries just over 512 bits, enough for the parent secret state (k_{par}, c_{par}) at 256 bits each. With a different primitive, the assignment should follow that primitive, and for HYDRA an asymmetry between the two slots decides it.

The two slots treat public constants differently: they are free in the input and costly in the key. A PRF adversary chooses the points at which it queries, so PRF security is a statement quantified over *all* input distributions. Restricting the queried inputs to a subset, even a small affine one, only weakens the adversary, and can never invalidate the assumption. The key slot behaves in the opposite way: PRF security is stated for a key drawn *uniformly* from $\mathbb{F}_p^4$, and a construction that draws its key from a proper subset is not covered by that statement. Filling key words with public constants therefore replaces the published assumption by a stronger, unpublished one about keys confined to an affine subspace. For HYDRA this substitution is particularly uncomfortable. Its key schedule $K' = K \,\|\, M_E K$ is *linear* by design (§A.6), which keeps it free in MPC, but it also carries an affine key space unchanged into every per-round key injection, and that is the kind of structure algebraic cryptanalysis exploits.

We therefore put all secret material in the key and all public material in the input. The key capacity is then fully used, no key word is constant, and domain separation sits in the input, next to the index it separates.

The re-assignment is not free. With the private key inside the PRF key, BIP32's additive rule $k_{child} = (I_L + k_{par}) \bmod n$ combines the PRF output with part of its own key, and the parent public key $k_{par} \cdot G$ is a function of that key as well. A security reduction to plain PRF security cannot simulate either, because it does not know the key. The proofs therefore use a further, explicitly stated assumption (Assumption 5.21), and Lemma 5.23 isolates what of it the published PRF claim already covers. Under BIP32's original assignment plain PRF security would suffice for this step, but only for a key confined to an affine subspace. The re-assignment thus trades one non-standard assumption for another; §5.8 states both.

4.2 Encodings and the derivation function

A derivation at index i evaluates the PRF as

$$E_{\mathsf{KeyEnc}(k_{par},\, c_{par})}(\mathsf{InEnc}(i)),$$

with two fixed, injective encodings into $\mathbb{F}_p^4$. (The name Derive is reserved for the whole procedure with its retries, Definition 5.2.)

The key encoding uses both parent secrets. Split each 256-bit secret into two 128-bit words, $k_{par} = k^{\mathrm{hi}} \,\|\, k^{\mathrm{lo}}$ and $c_{par} = c^{\mathrm{hi}} \,\|\, c^{\mathrm{lo}}$, and set

$$\mathsf{KeyEnc}(k_{par}, c_{par}) \;=\; (k^{\mathrm{hi}},\; k^{\mathrm{lo}},\; c^{\mathrm{hi}},\; c^{\mathrm{lo}}) \;\in\; \mathbb{F}_p^4.$$

Each word is $< 2^{128} < p$, so the embedding is lossless (§A.1), and the map is injective. No key word is a public constant: the key carries the whole parent secret state, just under 512 bits of entropy, against a target of $\kappa = 128$.

The input encoding contains public data only. The index i is read as an integer (it lies in $[2^{31}, 2^{32})$, so the hardened bit is its top bit), and

$$\mathsf{InEnc}(i) \;=\; (i,\; \delta_0,\; \delta_1,\; \delta_2) \;\in\; \mathbb{F}_p^4,$$

with $\delta_0, \delta_1, \delta_2 \in \mathbb{F}_p$ fixed public domain-separation constants. In HYDRA's terms (§2.2) the input is $[\nu \,\|\, IV]$ with nonce $\nu = i$ and fixed initial value $IV = (\delta_0, \delta_1, \delta_2)$, which is exactly the input

form for which HYDRA is specified. A key is evaluated on at most 2^{31} distinct inputs, far below the data limit $2^{\kappa/2} = 2^{64}$. The map is injective in i, which is all the security argument requires (§5.5): distinct hardened indices give distinct PRF inputs. The input contains no secret, so InEnc is evaluated in the clear.

The constants $\delta_0, \delta_1, \delta_2$ play no role in the security argument. Injectivity in i, requirement (R1) of Definition 5.1, is carried by the first input word alone, so the security reduction of §5 never uses $\delta_0, \delta_1, \delta_2$ and any fixed public values are admissible. Their only purpose is to fill the remaining three input words with a nothing-up-my-sleeve tag that domain-separates this derivation from any other use of a HYDRA key under the same parameters. We fix δ_0 to the integer whose little-endian byte string is the 14-byte ASCII string `BIP32-HYDRA-hd`, that is, $\delta_0 =$ `0x64682d41524459482d3233504942` (111 bits, so $\delta_0 < p$), and $\delta_1 = \delta_2 = 0$; a verifier reproduces the choice from the string. This is the one little-endian conversion of our construction; the serialization of §4.3 is big-endian, as in BIP32.

Hardened semantics are unchanged. Derivation still consumes k_{par}, so no path from the parent extended *public* key to a child private key exists, which is the defining property of hardened derivation and the reason BIP32 puts the private key into the computation at all.

Nor does the re-assignment change the cost. Key expansion, Even–Mansour pre-whitening, and per-round key injection are all linear and therefore local on shares (§2.3), so moving secrets between the key and input positions adds no Beaver triple. The encoding even becomes *cheaper*, since the input side is now entirely public, and the key side consumes $[\![k_{par}]\!]$ and $[\![c_{par}]\!]$ in the limb form they are already stored in (§4.4.1), so both encodings are free.

The 512-bit output $I = I_L \,\|\, I_R$ is obtained by serializing $E_K(\cdot)$ under the rejection rule of §4.3; then

$$k_{child} \;=\; (I_L + k_{par}) \bmod n, \qquad c_{child} \;=\; I_R,$$

subject to the validity checks $I_L < n$ and $k_{child} \neq 0$ of §4.4, exactly as in BIP32. Apart from the serialization of the field-valued output and its rejection rule (§4.3), everything after the PRF is BIP32's own: HYDRA takes the place of HMAC-SHA512. The parent key therefore enters twice: through the PRF key $\mathsf{KeyEnc}(k_{par}, c_{par})$ and as the additive offset.

Remark 4.1 (What the re-assignment buys, and where). Two distinct benefits, both used in §5. *(i) Uniform keys.* Under KeyEnc a uniform pair (k, c) on $[1, n-1] \times \{0,1\}^{256}$ gives a key uniform on the image of KeyEnc, a large proper subset of $\mathbb{F}_p^4$, and therefore within statistical distance $\varepsilon_{\text{enc}} < 2^{-117}$ of uniform on $\mathbb{F}_p^4$ (Lemma 5.22). The root pair is uniform by construction; the pair of a derived node is uniform once its parent's PRF has been replaced by a random function (Lemma 5.9), which is the step in which the proofs of §5 use it. The published HYDRA PRF assumption therefore applies directly to the derivation *without* BIP32's additive offset (Lemma 5.23). The offset and the public key need the additional Assumption 5.21. Under BIP32's original assignment the key would lie in a 2-dimensional affine subspace of $\mathbb{F}_p^4$, from which *no* statistical argument reaches the uniform case; a separate subspace-key assumption would be unavoidable there instead. *(ii) Partial-exposure standing.* If c_{par} becomes public (§5.7), BIP32's assignment makes the entire PRF key public and leaves no reduction at all, forcing an appeal to one-wayness of $m \mapsto E_K(m)$ under a *known* key. Under KeyEnc the same event leaves 256 bits of key secret, and the adversary's task becomes key recovery under partial key exposure (Assumption 5.27). That task is still a real cryptanalytic assumption about HYDRA, since the exposed limbs make K' affine in two unknowns, but it retains 256 secret bits where BIP32's assignment retains none. In this model the re-assignment is better; it does not remove the assumption.

In a threshold wallet the parent private key and chain code are secret-shared, $[\![k_{par}]\!]$ and $[\![c_{par}]\!]$, while the index i and the constants $\delta_0, \delta_1, \delta_2$ are public. Derivation runs the plain HYDRA PRF of §2.2 on the encoded shares:

$$[\![K]\!] \leftarrow \mathsf{KeyEnc}([\![k_{par}]\!], [\![c_{par}]\!]), \qquad m \leftarrow \mathsf{InEnc}(i), \qquad [\![E_K(m)]\!] \leftarrow \text{HYDRA}([\![K]\!], m).$$

The input m contains no secret at all, so it is a public $\mathbb{F}_p^4$ constant that every party holds identically and the pre-whitening $[\![m + K]\!] = m + [\![K]\!]$ is a local constant addition. Once encoded, the entire PRF evaluation proceeds inside $\mathbb{F}_p$ with the per-layer triple costs of §A.3, and the output is serialized with rejection (§4.3). All non-local cost of the wrapper therefore sits on the *output* side, in §4.4.

4.3 Fixed-width serialization

The PRF emits four elements of $\mathbb{F}_p$, which must be serialized into the fixed 512-bit output $I = (I_L \,\|\, I_R)$ expected by BIP32. Since $p = 2^{128} + 385$, a field element can lie in the overflow band $[2^{128}, 2^{128} + 384]$ and therefore does not always fit in a 128-bit word.

If any output word $\omega_l \geq 2^{128}$, the derivation is rejected and re-attempted with an incremented index, in direct analogy to BIP32's $I_L \geq n$ rule [29]. Otherwise each ω_l is encoded as a 16-byte big-endian string $\mathrm{ser}_{128}(\omega_l)$, leading zeros included, and

$$I \;=\; \mathrm{ser}_{128}(\omega_0) \,\|\, \mathrm{ser}_{128}(\omega_1) \,\|\, \mathrm{ser}_{128}(\omega_2) \,\|\, \mathrm{ser}_{128}(\omega_3) \in \{0,1\}^{512},$$

with I_L the first and I_R the last 32 bytes. This yields exactly $4 \times 128 = 512$ bits with no truncation and preserves the injectivity of the per-element encoding. The byte order is BIP32's: parse_{256}, which reads 32 bytes as a big-endian integer [29], gives $\mathrm{parse}_{256}(I_L) = \omega_0 \cdot 2^{128} + \omega_1$ and $\mathrm{parse}_{256}(I_R) = \omega_2 \cdot 2^{128} + \omega_3$, which are exactly the limb pairs (ω_0, ω_1) and (ω_2, ω_3) of §4.4.1. The child chain code I_R re-enters the next derivation through KeyEnc as the same limbs, so no byte string is ever formed inside the protocol: the arithmetic works on the limbs, and the byte string is the form in which a child is exported.

Rejection is rare. Per word, $\Pr[\omega_l \geq 2^{128}] = 385/p \approx 2^{-119.4}$, so per derivation call

$$\varepsilon_{\mathrm{ovf}} \;=\; 1 - \left(2^{128}/p\right)^4 \;\leq\; 4 \cdot 385/p \;\approx\; 2^{-117.4}.$$

In MPC the overflow test is a comparison on secret shares, and the same rejection structure recurs at the $I_L \geq n$ step for secp256k1 (§4.4), with $\varepsilon_{\mathrm{ord}} = 1 - n/2^{256} \approx 2^{-127.7}$. We write

$$\varepsilon_{\mathrm{stat}} \;=\; \varepsilon_{\mathrm{ovf}} + \varepsilon_{\mathrm{ord}} \;\approx\; 2^{-117.4} \;<\; 2^{-115}$$

for the combined quantity. The proofs of §5 work with the two rejection events separately (Lemma 5.5) and with the closely related $\varepsilon_{\mathrm{enc}} \leq \varepsilon_{\mathrm{ovf}} + \varepsilon_{\mathrm{ord}} + 2^{-256}$ (Lemma 5.22); $\varepsilon_{\mathrm{stat}}$ is the bound quoted for the per-attempt rejection probability (§6.3).

4.4 Mapping to secp256k1 scalars

The derivation PRF operates in $\mathbb{F}_p$, but a BIP32 child key is a secp256k1 scalar, i.e. an element of $\mathbb{Z}_n$ with n the (256-bit) group order. These are two different moduli, $p = 2^{128} + 385$ and $n \approx 2^{256}$, and every scalar-valued step of Algorithm 1 (ShareToWords, the addition modulo n, and the validity checks) lives at their boundary. This subsection fixes the scalar representation, specifies each of those steps on shares, and identifies the non-local work they need: bit decompositions, the adder, three comparisons with public constants and the zero test.

4.4.1 Scalar representation

We share all values over $\mathbb{F}_p$, so that the Hydra PRF runs natively (§2.3); a 256-bit scalar therefore cannot be a single field element. We represent it in *two 128-bit limbs, high limb first*: a scalar $x \in [0, 2^{256})$ is carried as an ordered pair $(x^{\mathrm{hi}}, x^{\mathrm{lo}})$ with $x = x^{\mathrm{hi}} \cdot 2^{128} + x^{\mathrm{lo}}$ and each limb in $[0, 2^{128})$. Since $2^{128} < p$, each limb embeds losslessly into $\mathbb{F}_p$ (§A.1), so the sharing $[\![x]\!] = ([\![x^{\mathrm{hi}}]\!], [\![x^{\mathrm{lo}}]\!])$ is a pair of ordinary $\mathbb{F}_p$-sharings.

The construction is closed under this representation. After the serialization rejection of §4.3 every output word $\omega_0, \ldots, \omega_3$ already lies in $[0, 2^{128})$, so

$$I_L = (\omega_0, \omega_1), \qquad I_R = (\omega_2, \omega_3)$$

are limb pairs with no conversion. The child chain code is $c_{child} = I_R$ directly, and the child private key $k_{child} = (I_L + k_{par}) \bmod n$ is produced in limb form by the addition of §4.4.2. Hence each derived key and chain code re-enters the next derivation already in canonical limb form, and under the key encoding of §4.2 the four limbs $(k^{\mathrm{hi}}_{child}, k^{\mathrm{lo}}_{child}, c^{\mathrm{hi}}_{child}, c^{\mathrm{lo}}_{child})$ *are* the next level's HYDRA key, with no repacking at all. Note the one distributional asymmetry this creates, which Lemma 5.22 accounts for: the chain-code limbs are uniform on $[0, 2^{128})^2$ whereas the key limbs encode a value uniform on $[1, n-1]$, so a key is $\varepsilon_{\mathrm{enc}}$-close to uniform on $\mathbb{F}_p^4$.

Because the canonical representation is already limb form, the key-encoding calls in Algorithm 1 are a regrouping of shares, not a conversion: SHARETOWORDS($[\![k_{par}]\!]$) returns the pair $([\![k^{\mathrm{hi}}]\!], [\![k^{\mathrm{lo}}]\!])$ already held, with zero communication, and likewise for $[\![c_{par}]\!]$.

4.4.2 The child-scalar computation

Given $I_L = (\omega_0, \omega_1)$ and the parent key $k_{par} = (k^{\mathrm{hi}}, k^{\mathrm{lo}})$, the construction sets, exactly as BIP32 (§2.1),

$$k_{child} = (I_L + k_{par}) \bmod n, \qquad \text{subject to} \qquad I_L < n \quad \text{and} \quad k_{child} \neq 0 \quad \text{(else reject)},$$

with n split into public limbs $n = (n^{\mathrm{hi}}, n^{\mathrm{lo}})$,

$$n^{\mathrm{hi}} = \texttt{0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE},$$
$$n^{\mathrm{lo}} = \texttt{0xBAAEDCE6AF48A03BBFD25E8CD0364141}.$$

Since $I_L < n$ and $k_{par} < n$, the sum $S = I_L + k_{par}$ lies in $[0, 2n)$, so the reduction is a single conditional subtraction of n, and $k_{child} = 0$ exactly when $S \in \{0, n\}$. The chain code I_R needs no processing: it is an opaque 256-bit value.

The addition and the comparisons have to work on bits. A field element carries no order, so $I_L \geq n$ cannot be decided by field arithmetic on shares, and the 256-bit value I_L does not even fit in one field element. Nor can the addition $I_L + k_{par}$ be done limb by limb in $\mathbb{F}_p$: native addition of two 128-bit limbs reduces modulo $p = 2^{128} + 385$, not modulo 2^{128}, so for $a, b \in [0, 2^{128})$ with $a + b \geq p$ the field result is $a + b - p$, which corrupts both the carry-out and, through the -385 offset, the low bits. We therefore bit-decompose the limbs of I_L and of k_{par} into 256 shared bits each, and do the addition and the comparisons as Boolean circuits on those bits. Appendix C gives the procedure (Algorithm 3) and the limb arithmetic behind it.

4.4.3 Rejection, cost, and scope

As in §4.3, the $I_L \geq n$ and $k_{child} = 0$ tests together open one public verdict bit and, on failure, advance the public counter of Algorithm 1. BIP32 discloses the same event by skipping the index. Both failures are astronomically rare: since $2^{256} - n \approx 2^{128.35}$, we have $\Pr[I_L \geq n] = \varepsilon_{\mathrm{ord}} \approx 2^{-127.7}$, and $\Pr[k_{child} = 0] = \Pr[I_L = n - k_{par}] = 2^{-256}$.

The wrapper's non-local cost is dominated by bit decomposition (of I_L and k_{par}, and in our implementation of all four output words, which the overflow test needs anyway), the 256-bit adder, three comparisons with public constants ($I_L \geq n$, $S \geq n$, $S^{\mathrm{lo}} < n^{\mathrm{lo}}$), and the zero test. Measured, this adds about 10,000 multiplications to the PRF's 147, so the wrapper dominates the multiplication count of a derivation, and about half of it is the addition modulo n. Its share of the running time depends on the protocol and the network (§7.3). Its circuits are shallow, while the PRF has the longest chain of dependent multiplications, so under network latency the

Algorithm 1 Hardened child derivation, executed by party P_j.

Input: Local shares $[\![k_{par}]\!]_j$, $[\![c_{par}]\!]_j$ of the parent private key and chain code; public hardened index idx $\in [2^{31}, 2^{32})$; public constants $\delta_0, \delta_1, \delta_2 \in \mathbb{F}_p$

Output: Local shares $[\![k_{child}]\!]_j$, $[\![c_{child}]\!]_j$ of the child private key and chain code, and the public accepted counter σ; or $\perp$ if the counter would reach 2^{32}

procedure HARDENEDDERIVE($[\![k_{par}]\!]_j$, $[\![c_{par}]\!]_j$, idx)
  ctr $\leftarrow$ idx ▷ *BIP32-style retry counter; public*
  repeat
    if ctr $\geq 2^{32}$ **then**
      return $\perp$ ▷ *index range exhausted; the request fails*
    Key encoding from both parent secrets (see §4.2)
    $([\![k^{\mathrm{hi}}]\!]_j, [\![k^{\mathrm{lo}}]\!]_j) \leftarrow$ SHARETOWORDS($[\![k_{par}]\!]_j$) ▷ *local regrouping of limbs; §4.4.1*
    $([\![c^{\mathrm{hi}}]\!]_j, [\![c^{\mathrm{lo}}]\!]_j) \leftarrow$ SHARETOWORDS($[\![c_{par}]\!]_j$) ▷ *local regrouping of limbs; §4.4.1*
    $\mathbf{K}_j \leftarrow ([\![k^{\mathrm{hi}}]\!]_j, [\![k^{\mathrm{lo}}]\!]_j, [\![c^{\mathrm{hi}}]\!]_j, [\![c^{\mathrm{lo}}]\!]_j)$ ▷ $[\![K]\!]_j \in \mathbb{F}_p^4$*; no constant key word*
    Input encoding, entirely public (see §4.2)
    $\mathbf{m} \leftarrow$ (ctr, δ_0, δ_1, δ_2) ▷ *public constant; held identically by every* P_j
    PRF evaluation on shares (the HYDRA *core, §2.2)*
    $[\![\boldsymbol{\omega}]\!]_j \leftarrow$ HYDRAPRF($\mathbf{K}_j$, $\mathbf{m}$) ▷ $[\![E_K(m)]\!]_j = ([\![\omega_0]\!]_j, \ldots, [\![\omega_3]\!]_j)$
    Serialization rejection (§4.3)
    ovf $\leftarrow \bigvee_{l=0}^{3}$ SECUREGE($[\![\omega_l]\!]_j$, 2^{128}) ▷ *secure comparisons; open the OR bit*
    if ovf $= 1$ **then**
      ctr $\leftarrow$ ctr $+ 1$; **continue** ▷ *overflow band; re-derive as in BIP32*
    BIP32 split and child validity (see §4.4)
    $[\![I_L]\!]_j \leftarrow ([\![\omega_0]\!]_j, [\![\omega_1]\!]_j)$ ▷ *limb pair; no conversion*
    $[\![k_{child}]\!]_j \leftarrow$ ADDMODN($[\![I_L]\!]_j$, $[\![k_{par}]\!]_j$) ▷ $(I_L + k_{par}) \bmod n$*; Alg. 3*
    bad $\leftarrow$ SECUREGE($[\![I_L]\!]_j$, n) $\vee$ SECUREISZERO($[\![k_{child}]\!]_j$) ▷ *BIP32 validity; open bit*
    if bad $= 1$ **then**
      ctr $\leftarrow$ ctr $+ 1$; **continue**
    $[\![c_{child}]\!]_j \leftarrow ([\![\omega_2]\!]_j, [\![\omega_3]\!]_j)$ ▷ $c_{child} = I_R$
  until a valid child is obtained
  return $([\![k_{child}]\!]_j, [\![c_{child}]\!]_j, \sigma = \mathrm{ctr})$ ▷ *the child is named* $w|\sigma$

PRF takes the largest share: about half of the time under malicious Shamir sharing at 100 ms, and over two thirds under semi-honest Shamir sharing. Without delay, most of the time goes to producing the preprocessed triples and random bits, nearly all of which the wrapper consumes. The cost is paid once per level. It comes from moving between the $\mathbb{F}_p$ domain of the PRF and the $\mathbb{Z}_n$ domain of the scalars, and it does not depend on which PRF is used.

This subsection produces k_{child} in limb form for further derivation. Threshold *signing*, which needs $\mathbb{Z}_n$ arithmetic, is orthogonal: the limb-form key would have to be converted to a $\mathbb{Z}_n$-sharing when signing is invoked, for example with mixed-circuit techniques such as edaBits [15]. We neither specify nor implement this conversion, and do not treat signing here.

4.5 The derivation procedure

Algorithm 1 gives the complete derivation from the view of one party P_j. HYDRAPRF($\cdot$) is the evaluation of E_K on shares (Appendix B). ADDMODN($\cdot$), SECUREGE($\cdot$) and SECUREISZERO($\cdot$) are the bit-level steps of §4.4 (Algorithm 3). Together with the overflow test they are the only non-local steps besides the PRF itself. Remark 4.2 below says what the rejections reveal.

Remark 4.2 (Rejection is public, matching BIP32). Each attempt opens at most two verdict bits,

the overflow verdict and, if there is no overflow, the validity verdict ($I_L \geq n$ or $k_{child} = 0$); the validity verdict is opened as a randomly masked product that is zero on rejection and nonzero otherwise except with probability $1/p$ when the mask is zero (Appendix C; Lemma 6.6). On rejection the public counter ctr advances, exactly as BIP32 skips an index. No secret-dependent value beyond these bits is revealed. BIP32 discloses that an index was skipped; our derivation discloses in addition which of the two tests rejected, and BIP32 has no overflow test. Both bits are part of the leakage λ that the games and the functionality of §6 account for, and each occurs with probability below 2^{-115}. As in BIP32, a rejected index is *invalid*: the child returned for a request at index idx is the child at the accepted counter value $\sigma \geq$ idx, it is recorded under the name $w|\sigma$ (§5.3), and no child exists at the skipped indices. A later request at any index between idx and σ therefore returns the same node, not a different one. If the counter would leave $[2^{31}, 2^{32})$, the request fails, as BIP32 has no further index to proceed to; this needs a rejection at the very end of the index range. An oblivious variant could be built if a stricter leakage profile is required: it would replace each opened bit by a shared select and run a fixed number of attempts, so that neither the verdicts nor the number of attempts are visible. We have not specified or implemented it.

5 Security of the derivation function

The goal of this section is Theorem 5.29. It says that the derivation of §4 gives what a wallet expects from hardened derivation. A hardened child is indistinguishable from a fresh uniform pair even when its siblings and their subtrees are revealed (sibling independence). A parent's private key cannot be computed by an adversary that holds its public key and the complete keys of all its children (parent–child hiding). The public keys of two children do not show whether they have the same parent (hardened-child unlinkability). Each bound is the offset-PRF advantage of Definition 5.4 times a factor linear in the depth of the target; parent–child hiding adds a discrete-logarithm term and 2^{-256} per derivation attempt at the target. We first fix an abstract derivation family and the advantage that the proofs consume (§5.1), and show that rejection leaves the accepted outputs uniform (§5.2). We then define the wallet tree and a single indistinguishability game on it (§5.3, §5.4). The path hybrid of Lemma 5.10 bounds this game by replacing the PRF with a random function at one node of the target path at a time, and the other notions follow from the same hybrid in §5.6. The rest of the section says what the model leaves out (§5.7), instantiates the assumptions with Hydra (§5.8), and points to the protocol-level statement of §6 (§5.9) before the theorem itself (§5.10).

Throughout, the guarantees of §5.6 and Theorem 5.29 rest on two assumptions: the *offset-PRF* property of the derivation family (Definition 5.4; for Hydra, Assumption 5.21) and hardness of discrete logarithm on secp256k1. The offset-PRF property is *not* the published PRF claim of Hydra. It is needed because in our construction the parent private key is part of the PRF key, is also BIP32's additive offset, and has a published public key. Lemma 5.23 shows what the published claim (Assumption 5.20) does cover, so that exactly two things are new: the key-dependent offset and the public key given as auxiliary input. A further assumption (Assumption 5.27) is needed only when a chain code is deliberately published (§5.7); it is used nowhere in Theorem 5.29.

Every result in this section is proved for an *abstract* derivation family. Hydra appears only in §5.8, where the assumptions are instantiated. The reductions never inspect F: they use it only through the advantage of Definition 5.4, so another PRF would change that one term. They are not, however, reductions to plain PRF security, for the reason just given. All games of this section are stand-alone; §5.9 explains how they carry over to the threshold protocol.

5.1 The abstract derivation function

The proofs work with an abstract form of the construction of §4: a derivation family (Definition 5.1), its derivation procedure with retries (Definition 5.2), and two advantages, plain PRF security (Definition 5.3) and the offset-PRF property that the reductions consume (Definition 5.4).

Definition 5.1 (Derivation family)**.** A *derivation family* is a tuple $\mathcal{D} = (F, \mathsf{KeyEnc}, \mathsf{InEnc}, \mathsf{Ser})$ consisting of a function family $F : \mathcal{K} \times \mathcal{X} \to \Sigma$ over arbitrary finite domains, encodings

$$\mathsf{KeyEnc} : [1, n-1] \times \{0,1\}^{256} \to \mathcal{K}, \qquad \mathsf{InEnc} : \mathcal{H} \to \mathcal{X},$$

and a serialization $\mathsf{Ser} : \Sigma \to \{0,1\}^{512} \cup \{\bot\}$, subject to three requirements:

(R1) *Input injectivity.* InEnc is injective.

(R2) *Key near-uniformity.* For (k, c) uniform on $[1, n-1] \times \{0,1\}^{256}$, the key $\mathsf{KeyEnc}(k, c)$ is within statistical distance ε_{enc} of uniform on $\mathcal{K}$.

(R3) *Clean serialization.* For σ uniform on Σ, the conditional distribution of $\mathsf{Ser}(\sigma)$ given $\mathsf{Ser}(\sigma) \neq \bot$ is uniform on $\{0,1\}^{512}$, and $\Pr[\mathsf{Ser}(\sigma) = \bot] \leq \varepsilon_{\text{ovf}}$.

For a function $f : \mathcal{X} \to \Sigma$, a pair (k, c) and an index $a \in \mathcal{H}$, the *attempt* at a computes $I_L \,\|\, I_R = \mathsf{Ser}(f(\mathsf{InEnc}(a)))$ and $k' = (I_L + k) \bmod n$. It is *accepted* if $\mathsf{Ser}(f(\mathsf{InEnc}(a))) \neq \bot$, $I_L < n$ and $k' \neq 0$, and its child is then (k', I_R).

Definition 5.2 (Derivation procedure)**.** For (k, c), a function $f : \mathcal{X} \to \Sigma$ and $i \in \mathcal{H}$, $\mathsf{Derive}^{f}_{k,c}(i)$ runs the attempts at $a = i, i+1, \ldots$ with f, recording for each the overflow verdict ($\mathsf{Ser}(\cdot) = \bot$) and, if there is no overflow, the validity verdict ($I_L \geq n$ or $k' = 0$). It stops at the first accepted attempt $a = \sigma$ and returns $(\sigma, (k', I_R), \lambda)$, where λ is the sequence of recorded verdict bits. If no attempt below 2^{32} is accepted, it returns $(\bot, \lambda)$. The real derivation is $\mathsf{Derive}_{k,c} = \mathsf{Derive}^{F(\mathsf{KeyEnc}(k,c),\,\cdot)}_{k,c}$; it is Algorithm 1 run in the clear.

BIP32 with HMAC-SHA512 is not a derivation family in this sense, because it places the private key in the PRF input; Remark 5.26 discusses it.

Definition 5.3 (PRF advantage)**.** For $F : \mathcal{K} \times \mathcal{X} \to \Sigma$ and an adversary $\mathcal{A}$ making at most q queries and running in time τ,

$$\mathrm{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}) \;=\; \Big|\Pr[\mathcal{A}^{F(K,\cdot)} = 1] - \Pr[\mathcal{A}^{R(\cdot)} = 1]\Big|,$$

with $K \leftarrow \mathcal{K}$ uniform and R a uniformly random function $\mathcal{X} \to \Sigma$. We write $\mathrm{Adv}^{\mathrm{prf}}_{F}(q, \tau)$ for the supremum over such adversaries.

Definition 5.4 (Offset-PRF advantage)**.** For a derivation family $\mathcal{D}$ and an adversary $\mathcal{A}$ making at most q oracle queries and running in time τ,

$$\mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}(\mathcal{A}) \;=\; \Big|\Pr[\mathcal{A}^{\mathsf{Derive}_{k,c}(\cdot)}(k \cdot G) = 1] - \Pr[\mathcal{A}^{\mathsf{Derive}^{R}_{k,c}(\cdot)}(k \cdot G) = 1]\Big|,$$

with (k, c) uniform on $[1, n-1] \times \{0,1\}^{256}$ and R a uniformly random function $\mathcal{X} \to \Sigma$. In the second experiment only F is replaced: the procedure still adds k to I_L and tests the result for zero. We write $\mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}(q, \tau)$ for the supremum over such adversaries.

Definition 5.4 is what the reductions below consume. It differs from Definition 5.3 in two ways: the adversary sees each output only after the parent key has been added to it, and it is given the parent public key. Both are forced by the construction. A reduction that embeds an unknown PRF key at a node must still produce that node's children, $(I_L + k) \bmod n$, and its public key $k \cdot G$, and with k inside the PRF key it cannot do either.

5.2 The rejection lemma

Lemma 5.5 is the only fact about rejection that the proofs need: conditioning a uniform value on acceptance leaves it uniform.

Lemma 5.5 (Rejection)**.** *Let X be uniform on $\mathbb{F}_p^4$ and $A = [0, 2^{128})^4$ the accept set of §4.3. Then:*

(i) the distribution of X conditioned on $X \in A$ is exactly uniform on A;

(ii) $\Pr[X \notin A] = \varepsilon_{\text{ovf}} = 1 - (2^{128}/p)^4 \leq 4 \cdot 385/p$*;*

(iii) for I_L uniform on $[0, 2^{256})$, the conditional distribution given $I_L < n$ is exactly uniform on $[0, n)$, and $\Pr[I_L \geq n] = \varepsilon_{\text{ord}} = 1 - n/2^{256}$.

Proof. (i) and (iii) are the observation that conditioning a uniform distribution on a subset yields the uniform distribution on that subset. (ii) is a direct count: $|A|/|\mathbb{F}_p^4| = (2^{128}/p)^4$, and $(1-x)^4 \geq 1 - 4x$. □

Parts (i) and (iii) are what Lemma 5.9 uses: accepted outputs are exactly uniform. This is the precise sense in which rejection is a branch-taken event rather than a distributional defect. Both rejection sites, output overflow in §4.3 and $I_L \geq n$ in §4.4, are covered by this single lemma. Both use the revealed model, in which the verdict bits and the resulting index are public, and in Theorem 5.29 they appear only as this public leakage.

5.3 The wallet tree

Let $\mathcal{H} = [2^{31}, 2^{32})$ be the hardened index set. A *node* is a path $w \in \mathcal{H}^*$; the root is the empty path ϵ. A wallet state assigns to each materialized node a pair $(k_w, c_w) \in [1, n-1] \times \{0,1\}^{256}$, with (k_ϵ, c_ϵ) sampled uniformly at setup. A derivation request (w, i) runs $\mathsf{Derive}_{k_w, c_w}(i)$; if it returns $(\sigma, (k', c'), \lambda)$, it materializes the node $w|\sigma$ with $(k_{w|\sigma}, c_{w|\sigma}) = (k', c')$. As in BIP32, the indices $i, \ldots, \sigma - 1$ are invalid at w and no node exists there, so every request at an index between i and σ materializes the same node. Write $\sigma(w, i)$ and $\lambda(w, i)$ for the returned index and verdict bits; both are public. Public keys are $P_w = k_w \cdot G$ on secp256k1. Remark 5.6 explains how this tree determines the proof technique.

Remark 5.6 (Hardened BIP32 is a GGM tree). Reading the recursion above with the assignment of §4.2, the parent secret state is the PRF *key*, the index is the PRF *input*, and half the output becomes the child's key material while the other half becomes the child's chain code. This is the shape of the GGM construction [18], generalized from a binary tree to arity 2^{31}, with BIP32's additive rule applied to the half that becomes the child's key. It fixes the proof technique: a hybrid that replaces the PRF by a random function one node at a time. Since revealed nodes off the challenge path cannot be replaced (their keys are known to the adversary), the hybrid runs along that path only, and the loss is a factor in its depth.

5.4 The core game

All notions of interest are corollaries of one indistinguishability game, which saves proving essentially the same hybrid several times.

Definition 5.7 (Game **HD-IND**$_{\mathcal{D}}$)**.** The adversary $\mathcal{A}$ first outputs a target path $w^* = (i_1, \ldots, i_d) \in \mathcal{H}^d$, $d \geq 1$; write $a_j = (i_1, \ldots, i_j)$ for $0 \leq j \leq d$, so $a_0 = \epsilon$ and $a_d = w^*$. The challenger samples (k_ϵ, c_ϵ) uniformly on $[1, n-1] \times \{0,1\}^{256}$ and $b \leftarrow \{0,1\}$, and gives $\mathcal{A}$ the following oracles.

- Expand(w, i): if w is materialized, $w \neq w^*$ and w is not a descendant of w^*, run the request (w, i) and return $(\sigma(w, i), \lambda(w, i))$.

- REVEAL(w): if w is materialized and $w \notin \{a_0, \ldots, a_d\}$, return (k_w, c_w).
- PUB(w): if w is materialized and $w \neq w^*$, return P_w.
- TEST: callable once. If w^* is materialized, return (k_{w^*}, c_{w^*}) if $b = 0$, and a fresh uniform element of $[1, n-1] \times \{0,1\}^{256}$ if $b = 1$; otherwise return $\perp$.

$\mathcal{A}$ outputs a guess b' and wins if $b' = b$. We set $\mathrm{Adv}^{\mathbf{HD}}_{\mathcal{D}}(\mathcal{A}) = |2\Pr[b' = b] - 1|$.

Remark 5.8 (Why the restrictions are what they are). *(i) No reveals on the challenge path.* F is a *public* function family; only the key is secret. An adversary holding (k_w, c_w) for an ancestor w of w^* therefore computes w^* itself. Reveals everywhere else, including siblings of path nodes and their whole subtrees, are permitted.

(ii) Public keys. PUB is available on every node except w^*: PUB(w^*) together with TEST would let the adversary compare P_{w^*} with the returned scalar. Public keys of challenges are handled by the other notions (see below). Public keys on the path are why Definition 5.4 gives the adversary $k \cdot G$.

(iii) No derivations at or below w^*. The children of w^* and their public keys are functions of w^*'s pair, and so are the verdict bits of a derivation under its key. Allowing such derivations would add a term that this section does not bound.

These are the standard conditions of a key-indistinguishability game: a public key or a derived child of the target is a function of the target, and would let the adversary test the challenge directly. Public keys are covered by the other notions: Proposition 5.13 gives the adversary the parent's public key together with all of its children, and in Proposition 5.18 the challenge itself consists of public keys. Lifting restriction (iii) would need one further PRF swap at every node at or below the target at which a derivation happens, together with a term for the verdict bits; we do not do this here.

Lemma 5.9 (Children of an ideal node)**.** *Fix* $(k, c) \in [1, n-1] \times \{0,1\}^{256}$ *and let* $R : \mathcal{X} \to \Sigma$ *be a uniformly random function. Across any sequence of requests to* $\mathsf{Derive}^R_{k,c}$*, the distinct children produced are mutually independent, each is uniform on* $[1, n-1] \times \{0,1\}^{256}$*, and together they are independent of* (k, c) *and of all returned verdict bits. The returned* σ *and* λ *depend on* k *only through events of probability at most* 2^{-256} *per attempt.*

Proof. Distinct children are accepted attempts at distinct indices, hence values of R at distinct inputs by (R1), so they are independent of each other and of the outputs of R at the rejected attempts. Fix an accepted attempt. By (R3) and Lemma 5.5, conditioned on acceptance of the serialization and on $I_L < n$, the value I_L is uniform on $[0, n)$ and I_R is uniform on $\{0,1\}^{256}$ and independent of it. For every fixed k, $k' = (I_L + k) \bmod n$ is then uniform on $\mathbb{Z}_n$, and conditioned on $k' \neq 0$ uniform on $[1, n-1]$. So the child is uniform on $[1, n-1] \times \{0,1\}^{256}$ whatever (k, c) is. The verdict bits depend on the outputs of R at the attempts, and on k only through the zero test, which fails exactly when $I_L = n - k$, an event of probability 2^{-256} for I_L uniform on $[0, 2^{256})$. This probability is over the attempt's output, not conditioned on $I_L < n$ (conditioned on it, it would be $1/n$). □

Lemma 5.9 is short, but every hybrid step below rests on it: once a node's PRF is replaced by a random function, its children are fresh uniform pairs, distributed exactly as the root. The work is done by the fresh random I_L; the additive offset is not needed for the children to be independent of the parent.

5.5 The path hybrid

Lemma 5.10 bounds the core game by replacing the PRF with a random function at one node of the target path at a time; Remark 5.11 explains why the bound is stated for selective targets.

Lemma 5.10 (Path hybrid)**.** *Let $\mathcal{A}$ be an adversary against* **HD-IND**$_{\mathcal{D}}$ *that names a target of depth d, makes at most q'* Expand *queries at any one node, and runs in time τ. Then*

$$\mathrm{Adv}^{\mathbf{HD}}_{\mathcal{D}}(\mathcal{A}) \;\le\; 2d \cdot \mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}(q', \tau'),$$

where τ' is τ plus the time to simulate the game.

Proof. For $0 \le j \le d$ let G_j be the game in which the derivations at $a_0, \ldots, a_{j-1}$ use independent uniformly random functions $R_0, \ldots, R_{j-1}$ in place of $F(\mathsf{KeyEnc}(k_{a_l}, c_{a_l}), \cdot)$, i.e. they run $\mathsf{Derive}^{R_l}_{k_{a_l}, c_{a_l}}$, and every other node uses F. G_0 is the real game.

Step $\mathsf{G}_{j-1} \to \mathsf{G}_j$. We build an adversary $\mathcal{B}_j$ against Definition 5.4. It receives $P = k \cdot G$ and an oracle O, which is either $\mathsf{Derive}_{k,c}$ or $\mathsf{Derive}^{R}_{k,c}$ for an unknown uniform (k, c). It runs $\mathcal{A}$, samples b itself, and receives w^*.

- If $j = 1$, the root is assigned the unknown pair (k, c). Otherwise $\mathcal{B}_j$ samples the root itself and runs the derivations at $a_0, \ldots, a_{j-2}$ with lazily sampled random functions, forming children with the offset from keys it knows. When the request at a_{j-2} that materializes a_{j-1} occurs, $\mathcal{B}_j$ computes that attempt's verdict bits as in G_{j-1}, but assigns a_{j-1} the unknown pair (k, c) instead of the child it computed. This does not change the distribution: in G_{j-1} that child is, by Lemma 5.9, uniform on $[1, n-1] \times \{0,1\}^{256}$ and independent of everything simulated so far, including the verdict bits.

- Expand(a_{j-1}, i) is answered with $O(i)$, and Pub(a_{j-1}) with P. Reveal(a_{j-1}) is not allowed by the game.

- Every other materialized node has a pair that $\mathcal{B}_j$ knows: the children of a_{j-1} come from O, nodes below them are computed with F, and all remaining nodes descend from keys $\mathcal{B}_j$ sampled. Their Expand, Reveal and Pub queries are answered directly.

- w^* is a descendant of a_{j-1}, so its pair is known; Test is answered according to b.

- *Consistency.* At every node, $\mathcal{B}_j$ records the outcome of each attempted index: a serialization rejection, a validity rejection, or an accepted child. A repeated request, or one whose retries run over indices already tried, reuses the recorded outcomes and the recorded child, so each index is queried at most once, at a_{j-1} to O, and the lazily sampled random functions are consistent. The pair of a_{j-1} is never known to $\mathcal{B}_j$; it is represented by P and O alone. Since the game exposes no raw PRF output words, $\mathcal{B}_j$ never needs the I_L or I_R from which that pair would have been formed.

$\mathcal{B}_j$ outputs 1 if $b' = b$. If O is real its simulation is exactly G_{j-1}, otherwise exactly G_j, and it makes at most q' queries to O. Hence $|\Pr_{\mathsf{G}_{j-1}}[b' = b] - \Pr_{\mathsf{G}_j}[b' = b]| \le \mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}(q', \tau')$.

The last game. In G_d, if w^* is materialized it is a child of a_{d-1} under R_{d-1}. By Lemma 5.9 its pair is uniform and independent of every other child and of all verdict bits, and the game gives the adversary no other function of it: w^* is neither revealed nor queried by Pub, and nothing is derived from it. The Test answer is therefore distributed identically for $b = 0$ and $b = 1$, jointly with the rest of the view, and $\Pr_{\mathsf{G}_d}[b' = b] = 1/2$. If w^* is not materialized, Test returns $\perp$ for both values of b.

Summing, $\mathrm{Adv}^{\mathbf{HD}}_{\mathcal{D}}(\mathcal{A}) = |2\Pr_{\mathsf{G}_0}[b' = b] - 1| \le 2\sum_{j=1}^{d} |\Pr_{\mathsf{G}_{j-1}}[b' = b] - \Pr_{\mathsf{G}_j}[b' = b]| \le 2d \cdot \mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}(q', \tau')$. □

No statistical terms appear in Lemma 5.10. Rejections are handled exactly: accepted children of a node whose PRF has been replaced by a random function are exactly uniform (Lemma 5.9), and rejected indices are invalid, as in BIP32, so the real and ideal games never differ because of a rejection.

Remark 5.11 (Adaptive targets). The game is *selective*: the adversary names w^* before its first query. For an adversary that chooses the target adaptively, the standard reduction guesses the target path among the nodes it could reach, which loses a factor up to $(q')^d$; polynomial-loss adaptive security of GGM-style trees is a known difficulty, see Fuchsbauer, Konstantinov, Pietrzak and Rao [16]. We claim only the selective bounds.

5.6 The wallet-level notions

The core game yields the three properties a wallet expects from hardened derivation: parent–child hiding (Definition 5.12, Proposition 5.13), sibling independence (Definition 5.15, Proposition 5.16) and hardened-child unlinkability (Definition 5.17, Proposition 5.18). Remark 5.14 explains why the first game gives the adversary the parent's public key, and Corollary 5.19 states what leaked nodes do not reveal.

Definition 5.12 (Parent–child hiding)**.** $\mathbf{PCH}_{\mathcal{D}}$: $\mathcal{A}$ first names a target node $w = (i_1, \ldots, i_d)$, $d \geq 0$. It has the oracles Expand (at every node), Pub (at every node, so in particular it obtains P_w) and Reveal, which is not allowed on w or on any ancestor of w. In particular it may reveal every child of w and every node below them. $\mathcal{A}$ wins if w is materialized and it outputs k_w.

Proposition 5.13. $\mathrm{Adv}^{\mathbf{PCH}}_{\mathcal{D}}(\mathcal{A}) \leq (d+1)\,\mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}(q', \tau') + \mathrm{Adv}^{\mathrm{dl}}_{\mathbb{G}}(\tau') + q_w \cdot 2^{-256}$, *where* $\mathrm{Adv}^{\mathrm{dl}}_{\mathbb{G}}$ *is the discrete-logarithm advantage on secp256k1,* q_w *is an upper bound on the number of attempts at* w *(in the simulated game without the zero test), and* q', τ' *are as in Lemma 5.10.*

Proof. Replace F by random functions at $a_0, \ldots, a_{d-1}$ and at w itself, one node at a time, as in Lemma 5.10. The step at w is where the auxiliary input is used: the reduction answers $\textsc{Pub}(w)$ with P and the derivations at w with its oracle. This is a search game, so each reduction can test whether $\mathcal{A}$ has won without knowing k_w, by comparing $\mathcal{A}$'s output against P_w; each step costs $\mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}$, for a total of $(d+1)$ of them. In the last game, by Lemma 5.9, the children of w are uniform and independent of k_w, everything revealed below them is a function of the children, and the view depends on k_w only through P_w and through the zero tests of the attempts at w. A discrete-logarithm solver given P therefore simulates this game with $P_w = P$, sampling w's children uniformly and computing w's verdict bits without the zero test. The simulation is exact unless one of the at most q_w zero tests at w would have failed, and it outputs $\mathcal{A}$'s guess. □

Remark 5.14 (Why the public key is in the game). It would be simpler to omit P_w, but a wallet that never publishes a parent public key is not a wallet. With P_w the statement matches deployment, at the cost of the discrete-logarithm term, which BIP32 with any PRF inherits. In our construction P_w costs more than that term: because k_w is part of the PRF key, it is also the reason Definition 5.4 gives the adversary $k \cdot G$.

In words, Proposition 5.13 says that the adversary may hold the *complete* extended private key of every child of w, and of every node below them, and still cannot compute k_w. As noted after Lemma 5.9, this comes from the fresh pseudorandom I_L of every child; the additive offset plays no part.

Definition 5.15 (Sibling independence)**.** $\mathbf{SI}_{\mathcal{D}}$: $\mathcal{A}$ first names a node w and an index i^*, and may reveal any children of w other than $w|i^*$, together with their subtrees. It must distinguish $(k_{w|i^*}, c_{w|i^*})$ from uniform. This is $\mathbf{HD\text{-}IND}_{\mathcal{D}}$ with $w^* = w|i^*$, and we write $\mathrm{Adv}^{\mathbf{SI}}_{\mathcal{D}}$ for its advantage. Since every target of **HD-IND** has this form, **SI** is not a new game: it is the wallet-level name of the core game, stated for the case a wallet cares about, a leaked sibling.

Proposition 5.16. $\mathrm{Adv}^{\mathbf{SI}}_{\mathcal{D}}(\mathcal{A}) \leq 2d \cdot \mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}(q', \tau')$, *where* d *is the depth of* $w|i^*$.

Proof. Immediate from Lemma 5.10: revealed siblings are off the path, which the game permits (Remark 5.8). □

Definition 5.17 (Hardened child unlinkability)**.** $\mathbf{HCU}_{\mathcal{D}}$: $\mathcal{A}$ first names two nodes $w_0 \neq w_1$ and indices $i_0 \neq i_1$. Let $u_0 = w_0|i_0$, $u_0' = w_0|i_1$ and $u_1 = w_1|i_1$ be the candidate challenge children; none of them may be w_0, w_1, or an ancestor of either. The challenger samples $b \leftarrow \{0,1\}$. $\mathcal{A}$ has Expand, Reveal and Pub as in Definition 5.7, except that Reveal is not allowed on w_0, w_1, their ancestors, or the candidate children, Pub is not allowed on the candidate children, and nothing is derived at or below them. A single Test query returns $\perp$ unless all three candidate children u_0, u_0' and u_1 are materialized, whatever the value of b. If they are, it returns $(P_{u_0}, P_{u_0'})$ if $b = 0$ (both children of w_0) and (P_{u_0}, P_{u_1}) if $b = 1$ (one child of each). $\mathcal{A}$ outputs b', and $\mathrm{Adv}^{\mathbf{HCU}}_{\mathcal{D}}(\mathcal{A}) = |2\Pr[b' = b] - 1|$.

Proposition 5.18. $\mathrm{Adv}^{\mathbf{HCU}}_{\mathcal{D}}(\mathcal{A}) \leq 2|U| \cdot \mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}(q', \tau') \leq 4d \cdot \mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}(q', \tau')$*, where U is the set of nodes on the paths from the root to w_0 and to w_1, both included, and d is the larger depth of the candidate children.*

Proof. Replace F by random functions at the nodes of U, parents first, exactly as in Lemma 5.10. No node of U is revealed, Pub on it is answered by the auxiliary input, and each step costs at most $2\,\mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}$ in the advantage. In the last game all three candidate children are children of w_0 or w_1 under random functions, so by Lemma 5.9 their private keys are independent and uniform on $[1, n-1]$, and independent of the rest of the view. Their public keys are then independent and uniform on the group without its identity. Whether Test returns $\perp$ depends only on which candidate children are materialized, which the Expand queries and their verdicts decide; neither depends on b. If Test returns a pair, both values of b yield two independent uniform points. In both cases $\Pr[b' = b] = 1/2$. Finally $|U| \leq 2d$. □

Corollary 5.19 (Leaked-node isolation)**.** *Let the adversary hold the complete pairs of any set L of nodes. Every node w^* of depth d such that no node of L is w^* or one of its ancestors remains indistinguishable from a fresh uniform pair, up to $2d \cdot \mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}$, provided, as in Definition 5.7, that P_{w^*} is not given and nothing is derived at or below w^*. In particular, a leaked child compromises neither its parent (Proposition 5.13) nor its siblings (Proposition 5.16).*

Proof. The nodes of L are off the path to w^*, so revealing them is allowed in **HD-IND** with target w^*; apply Lemma 5.10. □

5.7 Scope of the model

The games above are about the derivation function; corruption of the threshold parties is the subject of §6. Under threshold sharing the same access structure protects $[\![k_{par}]\!]$ and $[\![c_{par}]\!]$: a coalition that can reconstruct one can reconstruct the other, so no corruption of at most T parties yields the chain code alone. A chain code becomes public only through a deliberate act: the export of an extended public key (xpub).

The chain-code-exposure model is treated separately. Exporting an xpub publishes c_{par} while k_{par} stays secret. In that model the canonical argument for preferring hardened over non-hardened derivation, namely that a leaked child private key plus the parent xpub must not yield k_{par}, is *not* a consequence of PRF security, under any assignment. PRF security says outputs look random to someone who does not hold the key; it says nothing about hiding the key from someone who holds part of it. We therefore place this model outside Theorem 5.29 and treat it under the separate Assumption 5.27. This is a deliberate scoping decision: Assumption 5.27 is needed exactly when a node whose xpub is exported also has hardened children. In the common BIP44 layout the exported account xpub sits at the last hardened level, with only non-hardened children, so it is not in this regime; a wallet that exports the xpub of a node and also derives hardened children below that node is, and should be evaluated against Assumption 5.27.

The revealed-rejection model discloses the verdict bits of each derivation attempt, $\lambda(w, i)$, and hence the resulting public index $\sigma(w, i)$. This is written into $\mathcal{F}_{\mathsf{HD}}$ (Definition 6.3) and into

every game above rather than argued away, so the composed statements account for it. An oblivious variant (shared selects and a fixed number of attempts, §4.5) could remove the leakage at extra cost; it is not specified here and is not needed for any result.

5.8 Assumptions and instantiation with Hydra

Nothing above mentions Hydra. We now instantiate the requirements. Table 1 lists every assumption with its status, and Remark 5.28 compares them with those of HMAC-SHA512.

Assumption 5.20 (Hydra is a PRF)**.** For E_K as specified in §2.2 with the parameters of §2.4, restricted to inputs $[\nu \,\|\, IV]$ with the fixed $IV = (\delta_0, \delta_1, \delta_2)$ of §4.2 and $\nu \in \mathbb{F}_p$, the advantage $\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{Hydra}}(q, \tau)$ is negligible at the $\kappa = 128$ level for K uniform on $\mathbb{F}_p^4$ and $q \le 2^{\kappa/2} = 2^{64}$. This is the security claim of [20, §5.1, §5.5], used as published. Our use stays far inside it: one key is evaluated on at most 2^{31} inputs.

We read "negligible at the $\kappa = 128$ level", here and in Assumption 5.21, in the concrete sense of [20]: no adversary with $\tau < 2^{128}$ and the stated number of queries is expected to do noticeably better than generic key search. This is a hypothesis about the instance, supported by cryptanalysis, not a proved bound.

The strongest published analysis bearing on Assumption 5.20 is Steiner's Gröbner-basis cryptanalysis [28]. The designers set $R_H = \lceil 1.25 \cdot \max\{24, 2 + R_H^*\} \rceil$, a 25% margin over the smallest round number R_H^* that meets their security condition; with $R_H^* = 29$ this gives $R_H = 39$ [20, §5.5]. Steiner estimates that an ideal adversary (linear-algebra exponent $\omega = 2$) attacks 31 head rounds with about 126 bits of work and up to 33 rounds below 128 bits; his $R_H^* = 34$ implies $R_H = 45$ to keep the intended margin [28, §1.1]. This does not break 39 rounds, but the margin is smaller than intended. His estimates are for the prime $2^{127} + 45$ and rest on a computer-aided check of the instances he examined; we have not checked whether they carry over to our prime and matrices, in either direction. We keep $R_H = 39$ by design, so that Assumption 5.20 is the published claim at the designers' round numbers and not a claim at round numbers of our own. Nothing in the construction depends on the number of head rounds except the cost: a 45-round head would add 6 multiplications (153 instead of 147).

Assumption 5.21 (Hydra is an offset-PRF)**.** For the derivation family $\mathcal{D}_{\mathrm{Hydra}}$ of §4, with the encodings of §4.2 and the serialization of §4.3, the advantage $\mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}_{\mathrm{Hydra}}}(q, \tau)$ of Definition 5.4 is negligible at the $\kappa = 128$ level for $q \le 2^{31}$.

Assumption 5.21 is *not* a claim of [20], and it is not implied by Assumption 5.20 (Remark 5.24). It asks that Hydra remain pseudorandom when its output I_L is added to part of its own key and the result is tested for zero, and when the public key of that key part is known. The designers do not analyse key-dependent outputs of this kind, and they explicitly do not claim security against related-key attacks [20, §5.5]. They give two grounds for that: keys are sampled at random for each computation, so an attacker can neither know nor choose relations between them; and in an actively secure protocol a difference added to a shared key is detected. Neither ground covers our setting directly: derived child keys are not freshly sampled but computed from their parent, and the second ground needs active security (Remark D.1). We know of no attack that exploits either feature. Lemma 5.23 below isolates the part of Assumption 5.21 that the published claim covers.

Lemma 5.22 (Key distribution; discharges (R2))**.** *Let $K = \mathsf{KeyEnc}(k, c)$ as in §4.2 with (k, c) uniform on $[1, n-1] \times \{0,1\}^{256}$. This is the distribution of the root, and of a child of a node whose PRF is a random function (Lemma 5.9); for a derived node of the real tree it is not claimed, and the proofs use the lemma only in those two cases. Then the statistical distance between K and the uniform distribution on $\mathbb{F}_p^4$ is*

$$\varepsilon_{\mathrm{enc}} \;=\; 1 - \frac{(n-1) \cdot 2^{256}}{p^4} \;\approx\; 2^{-117.4},$$

and $\varepsilon_{\text{enc}} \le \varepsilon_{\text{ovf}} + \varepsilon_{\text{ord}} + 2^{-256}$.

Proof. KeyEnc is injective, and its image is a set of $(n-1)\cdot 2^{256}$ points of $\mathbb{F}_p^4$, on which K is uniform. The statistical distance between the uniform distribution on a subset S and the uniform distribution on $\mathbb{F}_p^4$ is $1-|S|/p^4$. For the bound, $(n-1)2^{256}/p^4 = (2^{128}/p)^4\cdot(n-1)/2^{256}$, and $1-xy \le (1-x)+(1-y)$ for $x, y \in [0,1]$. □

Lemma 5.23 (What the published claim covers)**.** *Let* Derive^0 *be the procedure of Definition 5.2 without the offset: the child key is* I_L *itself and the zero test is applied to* I_L*. Let* $\mathcal{A}$ *distinguish* $\mathsf{Derive}^0_{k,c}$ *from the same procedure with* F *replaced by a random function, with* (k,c) *uniform on* $[1, n-1]\times\{0,1\}^{256}$ *as in Definition 5.4, but* without *being given* $k\cdot G$*. If its queries cause at most* q_a *attempts in total in either experiment, then its advantage is at most* $\mathrm{Adv}^{\mathrm{prf}}_F(q_a,\tau') + \varepsilon_{\text{enc}}$.

Proof. $\mathsf{Derive}^0_{k,c}$ evaluates $F(\mathsf{KeyEnc}(k,c),\cdot)$ only at the public inputs $\mathsf{InEnc}(a)$, and computes everything it returns from those outputs by public operations that do not use (k,c). An adversary against it is therefore an adversary against F with key $\mathsf{KeyEnc}(k,c)$ making at most q_a queries. By (R2), that key is within ε_{enc} of uniform on $\mathcal{K}$, which Definition 5.3 requires. □

Comparing Lemma 5.23 with Definition 5.4, exactly two things are not covered by the published claim: the offset k in the child key and its zero test, and the auxiliary input $k\cdot G$.

Remark 5.24 (Plain PRF security does not imply the offset-PRF property). PRF security does not imply Definition 5.4. Let F' be a PRF keyed by the chain code alone, fix one input $x_0 = \mathsf{InEnc}(i_0)$, and define $F(\mathsf{KeyEnc}(k,c),x) = F'(c,x)$ for $x \ne x_0$, while at x_0 the output is serialized as $I_L = n-k$, with I_R taken from $F'(c,x_0)$. For uniform (k,c) this F is a PRF up to a statistical term: its outputs at $x \ne x_0$ depend only on c, and its output at x_0 is a function of k, which is uniform and independent of c. In the derivation, however, the attempt at i_0 yields $k' = (n-k+k) \bmod n = 0$ and is always rejected, whereas with a random function it is rejected with probability about 2^{-117}. The verdict bits distinguish the two.

Lemma 5.22 is where the re-assignment of §4.1 is used (Remark 4.1(i)): since no key word is a public constant, one statistical step reaches Assumption 5.20 (Lemma 5.23). Under BIP32's original assignment the key would be uniform on a 2-dimensional affine subspace, at statistical distance $1-p^{-2}$, essentially 1, from uniform on $\mathbb{F}_p^4$.

Remark 5.25 (A variant that needs only plain PRF security). The offset problem disappears if the private key never occupies the key slot. One way: take all 8 words of the HYDRA head output (§2.2) and use words 4–7 as the next level's PRF key, keeping (k_{child}, c_{child}) from words 0–3 as now. A reduction then knows each node's private key, and plain PRF security would suffice, with a statistical step for the key like that of Lemma 5.5 rather than Lemma 5.22, since the next key would be four output words, not $\mathsf{KeyEnc}(k,c)$. The head already computes all 8 words, so the variant needs no further multiplication; its cost is four more field elements of secret state per node. The reason we do not adopt it is the node state: a node would no longer be BIP32's pair (k,c), so extended keys could not be exported, stored or imported in BIP32's format. We keep BIP32's additive rule so that everything after the PRF is as BIP32 defines it; the pair format alone would not require it. With the rule comes the offset that Assumption 5.21 has to cover. Dropping the addition alone would not lead back to plain PRF security: the parent key would still be part of the PRF key while the adversary holds its public key, a case that Lemma 5.23 does not cover. Adopting the variant would change the derivation function, and with it the implementation, the test vectors and the benchmarks of §7; it is a design alternative, not a correction.

Remark 5.26 (BIP32 with HMAC-SHA512). BIP32 with HMAC-SHA512 is *not* a derivation family in the sense of Definition 5.1: InEnc takes only the index, whereas BIP32 places the private key in the HMAC input. The path hybrid of §5.5 applies to it as well, with the private key in the input rather than in the key. In BIP32 the chain code is the PRF key. At the

node where a reduction places its PRF oracle, the chain code is the oracle's unknown key and the private key is a value the reduction sampled itself, so it can form every HMAC input and the node's public key. Plain PRF security of HMAC-SHA512 keyed by the chain code would therefore suffice. We do not prove this here.

Assumption 5.27 (Partial key exposure; not used in Theorem 5.29)**.** Let a node's chain code c be published by an xpub export, while k stays secret. The adversary then holds c and $P = k \cdot G$, may request derivations at the node and see their verdict bits, and may obtain any of its children in full, i.e. pairs $((I_L + k) \bmod n,\ I_R)$ for indices of its choice. We assume that recovering k from this view costs 2^κ with $\kappa = 128$.

We do not state a weaker form, because the structure does not allow one. Exposure removes more than two of the eight words of the expanded key. Because the schedule $K' = K \,\|\, (M_E K)$ is *linear* (§A.6), fixing $(c^{\mathrm{hi}}, c^{\mathrm{lo}})$ makes the *entire* expanded key K' an affine function of the two remaining unknowns $(k^{\mathrm{hi}}, k^{\mathrm{lo}})$, so every per-round key injection depends on only those two field elements. Each revealed child gives one equation $I_L = k_{child} - k \bmod n$ in the same unknowns. Assumption 5.27 is therefore a bet on Hydra with a two-element unknown key at a round count calibrated for a uniform four-word key, and low-degree algebraic (Gröbner-basis) attacks target this regime. Steiner's analysis [28] shows that the quadratic Lai–Massey heads are the main target of such attacks even with a fully unknown key (see above); with two key words exposed the system to be solved is smaller still. It is not a claim of [20]; we state it as an open assumption. It is required only for the chain-code-exposure model of §5.7, and no step of Theorem 5.29 depends on it.

Remark 5.28 (Where Hydra and HMAC-SHA512 differ). At Assumption 5.27, substituting Hydra for HMAC-SHA512 has a cost. With SHA-512 one models the compression function as a random oracle and the property follows immediately. For Hydra there is no such shortcut. It is a low-degree algebraic design over $\mathbb{F}_p$ with a small round count, and under chain-code exposure its expanded key is affine in only two unknown field elements (Assumption 5.27); Gröbner-basis attacks exploit such systems [28]. We therefore cannot idealize the property, and state it as an open assumption instead.

This is also why Remark 4.1(ii) matters. Under BIP32's original assignment the same exposure makes the PRF key *fully* public, leaving no reduction and forcing an appeal to one-wayness of $m \mapsto E_K(m)$ under a known key. That assumption leaves no secret key material at all, whereas Assumption 5.27 still hides 256 bits; we give no formal reduction between the two.

More generally, the reductions of §5.6 are generic in F, but the *assumptions* do not become equal by that: "HMAC-SHA512 is a PRF" and "Hydra is an offset-PRF" are different and formally incomparable bets. The claim of this work is that the derivation layer can be proved once for any family in the sense of Definition 5.1, not that the two primitives are equally trusted.

5.9 Protocol-level statement

The games above are stand-alone: they concern the derivation *function* and cannot see the protocol. The protocol is the subject of a separate, simulation-based statement in §6. There, $\mathcal{F}_{\mathsf{HD}}$ (Definition 6.3) is a trusted party that stores every node's secrets, computes the derivation of §4 exactly, and leaks only the verdict sequence $\lambda(w, i)$ and the public commands. Theorem 6.7 shows that the protocol of Algorithms 1 and 3 UC-realizes $\mathcal{F}_{\mathsf{HD}}$, with no assumption on F, and Theorem 6.12 extends this, with abort, to malicious adversaries. Corollaries 6.9 and 6.14 are the only bridges between the two halves: they carry the bounds of the main theorem below to the real threshold wallet, against semi-honest and against malicious parties.

Because λ is part of $\mathcal{F}_{\mathsf{HD}}$ and of the games, the wallet-level guarantees hold for an adversary that sees the skip leakage.

Requirement	Status	Source	Consequence if it fails
PRF security of F, uniform key (Asm. 5.20)	Published claim	[20]; analysed in [28]: attacks reach 33 of 39 head rounds; [28] recommends 45	Lemma 5.23 fails, and in practice Asm. 5.21 with it; swap instantiation.
Offset-PRF with public key (Asm. 5.21)	New, not published	This work (Def. 5.4, Lem. 5.23, Rem. 5.24)	All of §5.6 fails.
(R2) Key near-uniform under KeyEnc	Proved	Lem. 5.22 (§5.8)	Not used by Thm. 5.29; only Lem. 5.23 fails.
(R1) Injectivity of InEnc	Proved	§4.2	Sibling independence fails.
(R3) Clean serialization	Proved (Lem. 5.5)	§4.3	Ideal outputs biased; all bounds shift.
Hardness of DL on secp256k1	Standard	Prop. 5.13	Parent key recoverable from its own public key.
Key recovery hard under chain-code exposure (Asm. 5.27)	Open	None	Only the xpub-export model of §5.7 is affected.
Secure realization of $\mathcal{F}_{\mathsf{HD}}$, semi-honest (Thm. 6.7)	Proved ($\mathcal{F}_{\mathsf{ABB}}$-hybrid; Cor. 6.8 for Shamir)	§6	Protocol-level leakage beyond λ.
Secure realization of $\mathcal{F}_{\mathsf{HD}}^{\mathrm{abort}}$, malicious (Thm. 6.12)	Proved ($\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$-hybrid)	§6.7	A deviating party could learn or change more than it can by aborting.
A protocol realizes $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$ (premise of Cor. 6.13)	Premise; not established here (Rem. 6.15)	[11, 26]	The malicious corollaries say nothing about that protocol; the other results stand.

Table 1: Assumptions consumed by the results of §5 and §6. Two entries are not published claims: Assumption 5.21, on which Theorem 5.29 rests, and Assumption 5.27, which is used only outside it. The last entry is a hypothesis of Corollary 6.13, not an assumption about an implementation.

5.10 Main theorem

Theorem 5.29 collects the results of this section in one statement.

Theorem 5.29 (Security of the derivation function)**.** *Let $\mathcal{D}$ be a derivation family satisfying (R1) and (R3) of Definition 5.1, and let $\mathcal{A}$ be an adversary that names a target of depth d,*

makes at most q' derivation requests at any one node, and runs in time τ. Then

$$\begin{aligned}\mathrm{Adv}^{\mathbf{HD}}_{\mathcal{D}}(\mathcal{A}),\ \mathrm{Adv}^{\mathbf{SI}}_{\mathcal{D}}(\mathcal{A}) &\le 2d \cdot \mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}(q', \tau'),\\ \mathrm{Adv}^{\mathbf{PCH}}_{\mathcal{D}}(\mathcal{A}) &\le (d+1) \cdot \mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}(q', \tau') + \mathrm{Adv}^{\mathrm{dl}}_{\mathbb{G}}(\tau') + q_w \cdot 2^{-256},\\ \mathrm{Adv}^{\mathbf{HCU}}_{\mathcal{D}}(\mathcal{A}) &\le 4d \cdot \mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}(q', \tau'),\end{aligned}$$

with q_w as in Proposition 5.13. For the construction of §4, all of these are negligible under Assumption 5.21 and hardness of discrete logarithm on secp256k1. The proofs use F only through $\mathrm{Adv}^{\mathrm{prf+}}_{\mathcal{D}}$.

Proof. Lemma 5.10 and Propositions 5.13, 5.16 and 5.18. □

In each bound d is the depth named in the corresponding statement (for **PCH** the depth of w, possibly 0; for **HCU** the larger depth of the candidate children). Requirement (R2) is not used in these bounds; it enters only Lemma 5.23.

In words: BIP32 hardened derivation with HYDRA in place of HMAC-SHA512 provides parent–child hiding, sibling independence and hardened-child unlinkability under Assumption 5.21. We do not prove these properties for HMAC-SHA512 (Remark 5.26), so the statement is about the new construction, not a comparison. The reductions are generic, but the assumption they consume is not plain PRF security: it is forced by keeping BIP32's additive rule while moving the private key into the PRF key (Remark 5.24). With BIP32's own assignment, plain PRF security would suffice (Remark 5.26), and Remark 5.25 describes a variant of this construction that needs only plain PRF security, at the cost of departing from BIP32's node state.

6 Threshold protocol and UC security

Section 5 proves properties of the derivation *function*. This section proves the complementary statement about the *protocol*: running the derivation on shares, as in Algorithms 1 and 3, reveals nothing beyond what a trusted party computing the same function would reveal: against *static, semi-honest* adversaries (Theorem 6.7), and, with abort, against *static malicious* adversaries (Theorem 6.12, §6.7). The two results are independent. No assumption about F, and no property of HYDRA, is used in this section; conversely, §5 never looks at the protocol. They are joined by Corollaries 6.9 and 6.14.

The proof is carried out in a hybrid model with an *arithmetic black box* $\mathcal{F}_{\mathsf{ABB}}$, an ideal functionality that stores values of $\mathbb{F}_p$ and computes on them on command. This choice keeps the proof independent of how the backend multiplies: Beaver triples, square pairs and Shamir resharing (§7.1, §7.8) are all realizations of the same command, and only the instantiations (Corollaries 6.8 and 6.13) depend on the choice.

6.1 Model

We use the universal composability (UC) framework [8]. An environment $\mathcal{Z}$ gives inputs to the parties $P_1, \ldots, P_N$ and reads their outputs, and an adversary $\mathcal{A}$ controls the corrupted parties. In the *real* execution the parties run the protocol Π_{HD} with access to $\mathcal{F}_{\mathsf{ABB}}$; in the *ideal* execution they forward their inputs to the functionality $\mathcal{F}_{\mathsf{HD}}$, and a simulator $\mathcal{S}$ plays the role of $\mathcal{A}$. The protocol *UC-realizes* $\mathcal{F}_{\mathsf{HD}}$ with error δ if for every $\mathcal{A}$ there is a $\mathcal{S}$ such that no $\mathcal{Z}$ distinguishes the two executions with advantage more than δ.

The adversary is *static* and *semi-honest*: it fixes a set $C \subseteq \{P_1, \ldots, P_N\}$ of corrupted parties before the execution starts, sees their entire view, and lets them follow the protocol. In the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model the theorem holds for every C; the bound $|C| \le T < N/2$ enters only when

$\mathcal{F}_{\mathsf{ABB}}$ is realized by Shamir sharing (Corollary 6.8). Malicious adversaries, which may deviate arbitrarily, are treated in §6.7; until then, "adversary" means a semi-honest one.

Every command a party receives from $\mathcal{Z}$ carries only public data: a node name w, a hardened index i, or a party index j. Secrets never enter from outside; the root is created by a setup command (Definition 6.1). We consider only environments that give *all parties the same commands in the same order*, as in a wallet whose parties execute the same program, and that issue the next command only after every honest party has produced its output for the previous one; call this class $\mathcal{Z}_{\text{sync}}$. The execution is thus sequential, with one outstanding command at a time, and a command is identified by its position in this sequence. The simulators below work online, one command at a time, so the views they produce agree with the real ones on every prefix of the execution. This is a restriction on the environment, not a property the protocol enforces: Theorem 6.7 is stated for $\mathcal{Z}_{\text{sync}}$, and composition with it applies to calling protocols that meet the same restriction.

In this section $[\![x]\!]$ means that x is stored in $\mathcal{F}_{\mathsf{ABB}}$ under some public handle. No party holds x or a share of it at this level of abstraction; shares reappear only in Corollary 6.8.

6.2 The arithmetic black box

The protocol computes on the arithmetic black box of Definition 6.1; Remark 6.2 explains why its commands cover every step of the derivation.

Definition 6.1 ($\mathcal{F}_{\mathsf{ABB}}$)**.** $\mathcal{F}_{\mathsf{ABB}}$ keeps a table Val from public handles to elements of $\mathbb{F}_p$. Each command below is executed when all parties send it with the same public arguments.

- $\textsc{Lin}(h, a_0, (a_j, h_j)_j)$: set $\mathsf{Val}[h] \leftarrow a_0 + \sum_j a_j \cdot \mathsf{Val}[h_j]$ for public $a_0, a_j \in \mathbb{F}_p$.
- $\textsc{Mult}(h, h_1, h_2)$: set $\mathsf{Val}[h] \leftarrow \mathsf{Val}[h_1] \cdot \mathsf{Val}[h_2]$.
- $\textsc{Rand}(h)$: set $\mathsf{Val}[h]$ to a fresh uniform element of $\mathbb{F}_p$.
- $\textsc{BitDec}(h, \ell, h_0, \ldots, h_{\ell-1})$: read $\mathsf{Val}[h]$ as an integer in $[0, p)$ and store its ℓ least significant bits under $h_0, \ldots, h_{\ell-1}$.
- $\textsc{Open}(h)$: send $\mathsf{Val}[h]$ to every party and to the adversary.
- $\textsc{OpenTo}(h, j)$: send $\mathsf{Val}[h]$ to party P_j only, and also to the adversary if P_j is corrupted.
- $\textsc{RootGen}(h_k^{\text{hi}}, h_k^{\text{lo}}, h_c^{\text{hi}}, h_c^{\text{lo}})$ (once): sample k_ϵ uniform in $[1, n-1]$ and c_ϵ uniform in $\{0,1\}^{256}$, and store their 128-bit limbs (§4.4.1) under the four handles.
- $\textsc{PubKey}(h^{\text{hi}}, h^{\text{lo}})$: compute $k = \mathsf{Val}[h^{\text{hi}}] \cdot 2^{128} + \mathsf{Val}[h^{\text{lo}}]$ and send the point $k \cdot G$ on secp256k1 to every party and to the adversary.

Apart from the values sent by Open and PubKey, and those sent by OpenTo to corrupted parties, the adversary learns only the sequence of commands, which is public.

Remark 6.2 (Why this black box). Every non-local step of the derivation is one of these commands. A Beaver multiplication, a product computed by local multiplication and resharing, and the product $r \cdot r^2$ in the cube step of Algorithm 2 are all Mult; a square pair (r, r^2) is Rand followed by Mult; the bit decompositions of §4.3 and §4.4 are BitDec; and the Boolean gates of the adder and the comparators are $x \wedge y = xy$ and $x \oplus y = x + y - 2xy$ on stored bits, i.e. Mult and Lin. RootGen and PubKey stand for master-key generation and threshold public-key computation, which this work does not treat (§6.8). PubKey is needed only so that public keys, which the notions of §5.6 give to the adversary, exist in the protocol at all.

6.3 The ideal functionality

The functionality is a trusted party that holds every node's secrets and computes the derivation of §4 *exactly*, including its retry rule, in the clear. It hands out no secret unless asked to.

Definition 6.3 ($\mathcal{F}_{\mathsf{HD}}$)**.** $\mathcal{F}_{\mathsf{HD}}$ is parameterized by the derivation of §4, i.e. the procedure $\mathsf{Derive}_{k,c}$ of Definition 5.2 for the family of §4. It keeps a table Tab from nodes to pairs $(k_w, c_w) \in [1, n-1] \times \{0,1\}^{256}$. On every command it receives, it first sends the command's name and its public arguments to the adversary.

- INIT: on (init) from all parties, if Tab is empty, sample k_ϵ uniform in $[1, n-1]$ and c_ϵ uniform in $\{0,1\}^{256}$, set $\mathsf{Tab}[\epsilon] \leftarrow (k_\epsilon, c_\epsilon)$, and output (init) to all parties. If Tab is not empty, leave it unchanged and output (error) to all parties.
- DERIVE: on (derive, w, i) from all parties, with $w \in \mathsf{Tab}$ and $i \in \mathcal{H}$, run $\mathsf{Derive}_{k_w, c_w}(i)$, i.e. Algorithm 1 in the clear. For each attempt it records the overflow verdict ovf and, if ovf $= 0$, the validity verdict bad; let $\lambda(w,i)$ be the resulting sequence of verdict bits and $\sigma(w,i) \in \mathcal{H}$ the counter value of the accepted attempt, as in §5.3. Send $\lambda(w,i)$ to the adversary. Following BIP32, the indices $i, \ldots, \sigma(w,i) - 1$ are invalid at w: set $\mathsf{Tab}[w|\sigma(w,i)]$ to the child and output $(\text{derived}, w, i, \sigma(w,i))$ to all parties. If no attempt below 2^{32} is accepted, output (failed, w, i) instead.
- OPEN: on (open, w) from all parties, with $w \in \mathsf{Tab}$, send (k_w, c_w) to all parties and to the adversary.
- EXPORT: on (export, w, j) from all parties, with $w \in \mathsf{Tab}$, send (k_w, c_w) to P_j only, and also to the adversary if P_j is corrupted.
- PUB: on (pub, w) from all parties, with $w \in \mathsf{Tab}$, send $P_w = k_w \cdot G$ to all parties and to the adversary.

A command whose node is not in Tab (in particular any command before INIT), or whose index is not in $\mathcal{H}$, is answered with (error) to all parties and changes nothing. $\mathcal{F}_{\mathsf{HD}}$ sends nothing else. Since DERIVE is deterministic given $\mathsf{Tab}[w]$, repeating a command recomputes the same entry.

Remark 6.4 (Three properties of $\mathcal{F}_{\mathsf{HD}}$). *(i) It computes the real function.* $\mathcal{F}_{\mathsf{HD}}$ evaluates Derive with F; it does not sample child keys at random. This is what keeps the present section free of any assumption on F: whether the function $\mathcal{F}_{\mathsf{HD}}$ computes is *secure* is the subject of the stand-alone games of §5, and Corollary 6.9 connects the two.

(ii) Its leakage is the verdict sequence and the public commands. The commands are public in the protocol too, since every party receives them; $\mathcal{F}_{\mathsf{HD}}$ forwards them so that the simulator learns them even when no party is corrupted. $\lambda(w,i)$ determines $\sigma(w,i)$, and adds to it only which of the two tests failed in each rejected attempt. For a random function each attempt is rejected with probability less than $\varepsilon_{\text{stat}} < 2^{-115}$ (Lemma 5.5); for HYDRA this is implied by Assumption 5.21. BIP32 itself discloses the skip (§2.1).

(iii) OPEN and EXPORT are the only ways a secret leaves. OPEN models a deliberate publication of a node, and corresponds to REVEAL in the game of Definition 5.7. PUB releases only the public key P_w, which the notions of §5.6 give to the adversary anyway. EXPORT delivers a node to a single party, e.g. the device of the wallet's owner, and reveals nothing to the adversary unless that party is corrupted. Our test programs (§7.1) open more than OPEN does: besides each child (k_{child}, c_{child}), which is an instance of OPEN, they also open the four PRF output words, which include I_L. Since $k_{child} - I_L \equiv k_{par} \pmod{n}$, those extra openings reveal the parent key. They exist only to compare the run with the reference implementation, and are outside Π_{HD} (§6.8).

6.4 The protocol

The protocol Π_{HD} (Definition 6.5) runs Algorithms 1 and 3 on the black box, and Lemma 6.6 shows that, unless a validity mask is zero, it stores and opens the same values as $\mathcal{F}_{\mathsf{HD}}$.

Definition 6.5 (Π_{HD})**.** All parties run the same public program in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model, and keep a public record of which nodes exist and under which handles their limbs are stored.

- Init: if no node is recorded, call RootGen and record ϵ, and output (init); otherwise output (error). Every party also outputs (error), without calling $\mathcal{F}_{\mathsf{ABB}}$, on a command whose node is not recorded or whose index is not in $\mathcal{H}$, exactly as $\mathcal{F}_{\mathsf{HD}}$ does; this check uses only the public record.
- Derive(w, i): run Algorithms 1 and 3 on the handles of w, with every step realized by $\mathcal{F}_{\mathsf{ABB}}$ commands as follows.
  - Every local step (KeyEnc, the public input InEnc, matrix products, round constants, key additions and injections, the Megafono sum, the feed-forward, truncation, bit recomposition and the conditional subtraction of n) is Lin.
  - Every Beaver mark of Algorithm 2 is Mult; each square pair is Rand followed by Mult; each Open mark of EmitCubes is Open of $s_l - r$.
  - BitDecompose is BitDec: with $\ell = 129$ for $\omega_0, \ldots, \omega_3$, and with $\ell = 128$ for $k^{\mathrm{hi}}, k^{\mathrm{lo}}$, once per level. The adder, the comparators, popcount and Hamming distance are Lin and Mult on the stored bits.
  - *Overflow verdict.* The OR of bit 128 of $\omega_0, \ldots, \omega_3$ is computed with Lin and Mult and opened; call it ovf.
  - *Validity verdict* (§C.1, implementation notes). With $v = (1 - [I_L \geq n]) \cdot \mathrm{popcount}(S) \cdot \mathrm{dist}(S, n)$, call Rand for ρ, compute ρv with Mult, and open it as y; the verdict is bad $= [y = 0]$.
  - On rejection, increment the public counter and repeat. On acceptance at the counter value σ, record the handles of the child limbs k^{hi}_{child}, k^{lo}_{child}, ω_2, ω_3 as the node $w|\sigma$, and output $(\mathrm{derived}, w, i, \sigma)$. If the counter would reach 2^{32}, output (failed, w, i).
- Open(w): call Open on the four handles of w and output the recomposed (k_w, c_w).
- Export(w, j): call OpenTo$(\cdot, j)$ on the four handles of w; P_j outputs the recomposed (k_w, c_w).
- Pub(w): call PubKey on the two key handles of w and output the point received.

Init, Derive and Open are the program that our derivation program (§7.1) compiles, with three differences. First, the implementation does not check the counter limit 2^{32}, which matters only after rejections at the very top of the index range. Second, the PRF program writes each PRF multiplication out as a Beaver step: it takes a preprocessed triple (a, b, ab), opens $x - a$ and $y - b$, and combines locally. In Π_{HD} this whole step is one Mult command, which the Beaver step realizes given ideal triples (Corollary 6.8); its two openings are part of that realization, not of Π_{HD}. Third, the derivation program contains the test switches of §7.1, which are not part of Π_{HD} (§6.8). Export and Pub are not implemented (§6.8).

Lemma 6.6 (Correctness)**.** *Call an attempt* faulty *if its validity test has $v \neq 0$ and $\rho = 0$. If no attempt is faulty, then every request (w, i) creates in Π_{HD} the same node $w|\sigma(w, i)$ as in $\mathcal{F}_{\mathsf{HD}}$, or fails in both; the values stored under its handles equal $\mathsf{Tab}[w|\sigma(w, i)]$, and the verdict bits opened during the request equal $\lambda(w, i)$.*

The proof, an induction over the nodes in the order of creation, is in Appendix D.

6.5 Security theorem

Theorem 6.7 states that, against static semi-honest adversaries, running Π_{HD} reveals nothing beyond what $\mathcal{F}_{\mathsf{HD}}$ reveals, up to a statistical error of q_{att}/p.

Theorem 6.7 (UC security of the threshold protocol)**.** *For environments in $\mathcal{Z}_{\mathrm{sync}}$ (§6.1), Π_{HD} UC-realizes $\mathcal{F}_{\mathsf{HD}}$ in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model against static semi-honest adversaries corrupting any set of parties, with statistical error at most q_{att}/p, where q_{att} is an upper bound on the total number of attempts that reach the validity test in the ideal execution (more sharply, the error is at most $\mathbb{E}[Q]/p$ for the random number Q of such attempts). No computational assumption is used.*

Proof sketch. The simulator receives the verdict sequence $\lambda(w, i)$ and the public commands from $\mathcal{F}_{\mathsf{HD}}$ and follows the public program. A derivation attempt opens three kinds of values. Each cube opening $s_l - r$ is masked by a fresh random r, none of whose other uses is opened, so it is uniform on $\mathbb{F}_p$ and the simulator samples it. The overflow opening is a bit of λ. The validity opening is ρv for a fresh random ρ: it is 0 when the attempt is rejected and uniform on $\mathbb{F}_p \setminus \{0\}$ otherwise, unless $\rho = 0$, which happens with probability $1/p$ per attempt and accounts for the error term. The values opened by Open, Export and Pub are given to the simulator by $\mathcal{F}_{\mathsf{HD}}$. Appendix D gives the simulator and the four hybrids in full. □

Theorem 6.7 says that, apart from the verdict bits, everything a corrupted party sees is either public by design or uniform and independent of the secrets. Since the proof uses no property of F, it holds for any function family computed by a straight-line program of $\mathcal{F}_{\mathsf{ABB}}$ commands, and in particular for Hydra with any choice of round numbers.

6.6 Instantiation and composition

Two corollaries connect Theorem 6.7 to the implementation and to the games of §5: Corollary 6.8 realizes the black box with semi-honest Shamir sharing, and Corollary 6.9 carries the bounds of Theorem 5.29 to the real wallet.

Corollary 6.8 (Semi-honest Shamir instantiation)**.** *Let N parties share values over $\mathbb{F}_p$ with Shamir's scheme [27] of threshold $T < N/2$, and realize the commands of $\mathcal{F}_{\mathsf{ABB}}$ as follows: Lin locally; Mult by local multiplication and resharing [7], or by Beaver triples [6] that are themselves produced by local multiplication and resharing; Rand by summing random sharings contributed by the parties (our implementation takes the first component of a fresh triple, which is such a value); Open by reconstruction; OpenTo$(\cdot, j)$ by sending the shares to P_j only; and BitDec from shared random bits and one masked opening [11, 15]. The parties communicate over ideal pairwise private and authenticated channels; with encrypted channels, as in a deployment, the security of the channel encryption is an additional computational assumption, not covered by the statistical bound below. RootGen and PubKey are replaced by two share-level ideal functionalities, which this work does not realize: $\mathcal{F}_{\mathrm{root}}$ samples (k_ϵ, c_ϵ) uniformly on $[1, n-1] \times \{0,1\}^{256}$ and deals to every party its Shamir shares (of threshold T, fresh random polynomials) of the four limbs, and $\mathcal{F}_{\mathrm{pub}}$ receives every party's shares of the two key limbs, reconstructs k, and sends $k \cdot G$ to every party and to the adversary. In the $(\mathcal{F}_{\mathrm{root}}, \mathcal{F}_{\mathrm{pub}})$-hybrid model, the resulting protocol UC-realizes $\mathcal{F}_{\mathsf{HD}}$, for environments in $\mathcal{Z}_{\mathrm{sync}}$, against static semi-honest adversaries corrupting at most T parties, with statistical error at most*

$$\delta_{\mathrm{UC}} \;=\; \frac{q_{\mathrm{att}}}{p} \;+\; q_{\mathrm{bd}} \cdot \frac{385}{p},$$

where q_{bd} is an upper bound on the number of bit decompositions of full field elements that mask with a 128-bit random value, as in our implementation. In our implementation every bit decomposition is of this kind, including those of the limbs $k^{\mathrm{hi}}, k^{\mathrm{lo}}$, so $q_{\mathrm{bd}} = 4 \cdot (\text{number of attempts}) + 2 \cdot (\text{number of derivations})$, with both numbers read as upper bounds.

Corollary 6.9 (Wallet guarantees for the real protocol)**.** *The games of Theorem 5.29 are standalone. Every bound of Theorem 5.29 also holds for the threshold wallet that runs* Π_{HD} *under semi-honest Shamir sharing, against an adversary that corrupts at most* T *parties and names its target first, as the selective games require, with the running time increased by that of* $\mathcal{S}$ *and with an additive loss of at most* $2\delta_{\mathrm{UC}}$ *for the decisional notions* **HD-IND**, **SI** *and* **HCU** *(and Corollary 5.19), and of at most* δ_{UC} *for the search notion* **PCH**, *where* $\delta_{\mathrm{UC}} = q_{\mathrm{att}}/p + q_{\mathrm{bd}} \cdot 385/p$ *is the error of Corollary 6.8.*

Both corollaries are proved in Appendix D. In the composition proof the environment obtains secret values of the games through Export to an honest party. The contents of such an export are hidden, but the command itself is public, so the environment issues the same exports for both values of the challenge bit; the factor 2 comes from converting a winning probability into the advantage $|2\Pr[b' = b] - 1|$. For **PCH** the environment instead records P_w with $\textsc{Pub}(w)$ when w is created and checks the final guess locally, so no command follows the adversary's output. Corollary 6.9 joins the two halves of the security argument. Neither uses the other (Theorem 6.7 never looks at F, and §5 never looks at the protocol), so an improved bound in §5 carries over unchanged to the real wallet.

The level $\kappa = 128$ refers to the computational assumptions on Hydra (Assumptions 5.20 and 5.21). The statistical terms are separate from it, and they are distances, not attacks of the corresponding cost. The key-encoding distance $\varepsilon_{\mathrm{enc}} \approx 2^{-117.4}$ enters only Lemma 5.23, not Theorem 5.29. The protocol error of Corollary 6.8 is $(1 + 6 \cdot 385)/p \approx 2^{-116.8}$ for one derivation with one attempt, and it grows linearly with the number of attempts and of derivations.

6.7 Malicious adversaries

Theorem 6.7 covers parties that follow the protocol. Das et al. [13] argue that hardened derivation must withstand parties that deviate from it, and our benchmarks include malicious Shamir sharing (§7). We now treat *static malicious* adversaries, which may deviate arbitrarily, with the standard relaxation of *security with abort*: a deviating party can make the execution stop, but cannot learn or change anything else. The argument has the same two layers as before. In a hybrid model with a black box that aborts on deviation (Definition 6.10), the protocol is proved to realize the derivation functionality with abort (Definition 6.11 and Theorem 6.12). The result then holds for any protocol that realizes such a black box (Corollary 6.13). Which protocols do, and how far the implementation we measure is covered, is discussed after it (Remark 6.15).

Definition 6.10 ($\mathcal{F}_{\mathsf{ABB}}$ with abort)**.** $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$ is $\mathcal{F}_{\mathsf{ABB}}$ (Definition 6.1) with two changes.

- A command is executed only if every party sends it with the same arguments. If a corrupted party sends a different command or different arguments, $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$ sends $\perp$ to every party and halts. Until every party has sent the command, $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$ waits, so a corrupted party that withholds it delays the execution for as long as it withholds it.
- Every value it releases (Open, PubKey, and OpenTo to a corrupted party) is sent to the adversary first. The adversary then either lets the command complete, or sends abort, upon which $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$ sends $\perp$ to every party and halts. It may send abort after any command.

A party of Π_{HD} that receives $\perp$ from $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$ outputs $\perp$ and halts.

Definition 6.11 ($\mathcal{F}_{\mathsf{HD}}$ with abort)**.** $\mathcal{F}_{\mathsf{HD}}^{\mathrm{abort}}$ is $\mathcal{F}_{\mathsf{HD}}$ (Definition 6.3) with one change. For every command, it first computes the command and sends the adversary everything $\mathcal{F}_{\mathsf{HD}}$ would give it: the command's name and public arguments, the verdict sequence $\lambda(w, i)$, and the values released by Open, Pub, and Export to a corrupted party. It then waits for the adversary's answer, with no time limit; until the answer arrives, the honest parties receive no output. On continue it updates Tab and gives the honest parties their outputs as $\mathcal{F}_{\mathsf{HD}}$ does; on abort it sends

$\perp$ to every party and halts. The adversary may also send abort at any other time, in particular between two commands, with the same effect.

Theorem 6.12 (Security against malicious adversaries, with abort)**.** *For environments in $\mathcal{Z}_{\text{sync}}$, Π_{HD} UC-realizes $\mathcal{F}_{\mathsf{HD}}^{\text{abort}}$ in the $\mathcal{F}_{\mathsf{ABB}}^{\text{abort}}$-hybrid model against static malicious adversaries corrupting any set of parties, with statistical error at most q_{att}/p, with q_{att} as in Theorem 6.7. No computational assumption is used.*

Proof sketch. In the $\mathcal{F}_{\mathsf{ABB}}^{\text{abort}}$-hybrid model every value of the derivation lives inside the black box, and the parties have no private inputs: the root comes from RootGen, and every command carries only public data. A corrupted party that sends anything other than the commands of Π_{HD} causes an abort, and one that withholds a command delays the execution. So a malicious adversary can do three things beyond what a semi-honest one can (deviate, abort, or withhold a command), and each of them only stops or delays the execution for every party. The simulator runs the simulator of Theorem 6.7 command by command, checks the commands the adversary sends for the corrupted parties, forwards an abort to $\mathcal{F}_{\mathsf{HD}}^{\text{abort}}$ at the first deviation, and holds back its answer to $\mathcal{F}_{\mathsf{HD}}^{\text{abort}}$ while a command is withheld. Up to the first deviation or abort both executions are executions of Π_{HD} in which every party follows the protocol, possibly with delays, and whether and when the adversary deviates, aborts or withholds is a function of its view. The full proof is in Appendix D. □

In the black-box model a malicious party has no inputs to lie about and no shares to corrupt, so its only power is to stop or delay the execution. All the difficulty of malicious security lies in realizing $\mathcal{F}_{\mathsf{ABB}}^{\text{abort}}$, and that is the premise of the next corollary.

Corollary 6.13 (Instantiation with abort)**.** *Let Π_{ABB} be a protocol for N parties that UC-realizes $\mathcal{F}_{\mathsf{ABB}}^{\text{abort}}$, for environments in $\mathcal{Z}_{\text{sync}}$, against static malicious adversaries corrupting at most T parties, with statistical error at most ε_{ABB}. Then Π_{HD}, with every call to $\mathcal{F}_{\mathsf{ABB}}^{\text{abort}}$ replaced by a call to Π_{ABB}, UC-realizes $\mathcal{F}_{\mathsf{HD}}^{\text{abort}}$, for environments in $\mathcal{Z}_{\text{sync}}$, against static malicious adversaries corrupting at most T parties, with statistical error at most $q_{\text{att}}/p + \varepsilon_{\text{ABB}}$.*

Corollary 6.14 (Wallet guarantees against malicious parties)**.** *Let Π_{ABB} and ε_{ABB} be as in Corollary 6.13. Every bound of Theorem 5.29 also holds for the threshold wallet that runs Π_{HD} over Π_{ABB}, against an adversary that corrupts at most T parties, lets them deviate arbitrarily, and names its target first, with the running time increased by that of $\mathcal{S}^{\text{abort}}$ and with an additive loss of at most $2(q_{\text{att}}/p + \varepsilon_{\text{ABB}})$ for the decisional notions and $q_{\text{att}}/p + \varepsilon_{\text{ABB}}$ for* **PCH**. *A run that aborts, or that stalls because a corrupted party withholds a command (including the Pub(w) command issued when w is materialized in* **PCH** *and the Export command that answers Test in the decisional notions), counts as a random guess for the decisional notions and as a failure for* **PCH***; this loses nothing, since an adversary can stop its queries instead.*

Corollary 6.13 follows from Theorem 6.12 and the UC composition theorem [8]. Composition is used only with callers that meet the restriction of §6.1: Π_{HD} issues the commands of the black box to all parties in the same order, one at a time. Corollary 6.14 is proved in Appendix D. Neither corollary depends on a particular implementation.

Remark 6.15 (The premise, and the implementation we measure)*. Published protocols.* For $T < N/2$, the protocols of Lindell and Nof [26] provide the operations of the Rand, Lin, Mult, Open and OpenTo commands, secure with abort against static malicious adversaries, also with Shamir sharing [26, Theorem 4.3, Appendix B], and are universally composable by [25] since all inputs are fixed in advance [26, Appendix A]. Damgård et al. [11] realize BitDec on top of a black box with these commands, also against malicious adversaries; each random bit it uses is obtained by opening the square of a random shared element and dividing the element by a public square root of the opened value. RootGen and PubKey are realized, as in Corollary 6.8, by the share-level functionalities $\mathcal{F}_{\text{root}}$ and $\mathcal{F}_{\text{pub}}$, here in versions that check the shares they receive

and abort on inconsistency. The cited protocols thus provide the ingredients of a protocol Π_{ABB}. We do not show here that their composition realizes $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$. Two points would need an argument: Lindell and Nof let the adversary decide for each honest party whether it aborts, whereas $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$ aborts for all parties; and their theorem is stated for a fixed arithmetic circuit, whereas $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$ keeps state, opens values during the computation and continues depending on them. Corollary 6.13 therefore stays conditional on such a realization.

The implementation we measure. The malicious Shamir protocol of MP-SPDZ [22], version 0.2.8 (`malicious-shamir-party.x`), which we benchmark in §7, follows Lindell and Nof for triples, multiplication and openings: triples are produced semi-honestly and checked by sacrificing, and every opening is checked for consistency. Its random bits are produced differently: each is the XOR of bits chosen by $T + 1$ parties, computed with semi-honest multiplications, and is then checked only for being a bit, by sacrificing it against a square (`MaliciousBitOnlyRepPrep::buffer_bits` in `Protocols/MaliciousRepPrep.hpp`, which takes the bits from `BitPrep::buffer_ring_bits_without_check` in `Protocols/ReplicatedPrep.hpp`). A corrupted party may be able to add an error to one of these multiplications that leaves a bit for one value of the result and not for the other, so that whether the check passes depends on the bit. These bits mask the bit decompositions of k_{par} and of the four output words, so such a selective failure would leak information about k_{par} and the child. We have not tested this attack, and we do not claim that the implementation meets the premise of Corollary 6.13. Our malicious Shamir figures are measurements of this implementation; an implementation with random bits produced as in [11] differs only in that step, and we leave it to future work. MP-SPDZ's option to produce bits from opened squares (`-Q`) is not used by its malicious Shamir protocol: with it, a derivation has the same rounds, data and triples as without it.

6.8 What the theorems cover

- *Covered.* The protocol of Algorithms 1 and 3, with HYDRA evaluated as in Appendix B, any number of derivations along any paths, retries, and deliberate export of nodes, against semi-honest adversaries (Theorem 6.7) and, with abort, against malicious ones (Theorem 6.12).

- *Implementation of malicious security.* The malicious corollaries hold for any protocol that realizes $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$. We do not show that MP-SPDZ's malicious Shamir protocol, which we benchmark, is one (Remark 6.15).

- *Master-key generation* is modelled by the ideal ROOTGEN, and at the share level by $\mathcal{F}_{\mathrm{root}}$ (Corollary 6.8). Realizing it requires either sampling the root inside MPC or deriving it from a seed, and is outside this work.

- *Public keys and signing.* Public keys are modelled by the ideal command PUBKEY of $\mathcal{F}_{\mathsf{ABB}}$, at the share level by $\mathcal{F}_{\mathrm{pub}}$, so that the notions of §5.6, which give public keys to the adversary, carry over (Corollary 6.9). Computing $P_w = k_w \cdot G$ on shares is not implemented, and its protocol is not covered here. Signing is neither modelled nor implemented (§4.4.3).

- *Test harness.* The simplifications of §7.1 (the parent key fixed at compile time, the test switch that forces rejections, and the opening of the PRF output words) are not part of Π_{HD}. The opening of each child (k_{child}, c_{child}) is an instance of OPEN; the opening of the PRF output words is not, since together with the child it reveals the parent key (Remark 6.4).

- *Adaptive corruption* is not considered, and neither are environments outside $\mathcal{Z}_{\mathrm{sync}}$.

- *Availability.* The theorems are about privacy and correctness. They do not remove the requirement of Remark 7.1 that all N parties take part in every run, and with abort a single malicious party can stop any run, by aborting or by withholding a command; it can never make an honest party accept a wrong key. The ideal derivation is deterministic for a fixed parent and index. A deployment may run an aborted derivation again from the same wallet state with fresh preprocessing; the security of such restarts needs a model with persistent state and is outside our results.

7 Implementation and evaluation

This section compares the construction with the mechanism it replaces: BIP32 hardened derivation with HMAC-SHA512 [24], evaluated on secret shares in the same MPC setting.

7.1 Implementation

The protocol of §4 is realized by two MP-SPDZ [22] programs. `bip32-hydra.mpc` evaluates the Hydra PRF on shares. `poc_derive.mpc` runs that PRF code unchanged inside the full derivation of Algorithm 1: the overflow test on shares, the addition $k_{child} = (I_L + k_{par}) \bmod n$ on shares, the validity test, the retry loop, and the output $(k_{child}, c_{child} = I_R)$, with each child fed back as the next level's parent. Its outputs, including the child keys, agree with an independent cleartext implementation of Derive on every test case, including forced retries and three-level paths, and are identical under semi-honest and malicious 5-party Shamir sharing ($T = 2$) and 2-party MASCOT. Three simplifications serve testing only: the parent key is supplied when the program is compiled rather than input privately by each party; the PRF output words, the child key and the chain code are opened at the end of each level so they can be compared with the reference; and a test switch can force the first attempts to be rejected, so that the retry path actually runs. Supplying the key at compile time does not let the compiler simplify the work: the program loads it into secret shares before any computation, and the compiler counts the 147 multiplications of §2.3 for the PRF and 10,120 triples for the whole derivation. We also ran the limb arithmetic of Appendix C on shares for 32 chosen pairs (I_L, k_{par}), among them $I_L = n$, $I_L = 2^{256} - 1$ with $k_{par} = n - 1$, sums $I_L + k_{par}$ equal to n and to $2n - 2$, and a case with a borrow between the limbs; verdicts and child keys agree with BIP32's rule in every case. Our implementation builds the random bit decompositions it needs from individually generated shared random bits; MP-SPDZ's edaBit option [15] gave the same outputs but was about five times slower under Shamir sharing. The programs, the cleartext implementation, the test vectors, the benchmark scripts and the recorded runs are available at `https://github.com/CemreOzdogan/bip32-hydra` (commit `604218b0db1c`), together with a Dockerfile that builds the MP-SPDZ version we used.

MP-SPDZ keeps two phases strictly apart. The compiler turns the program into a fixed list of instructions once, before any party, key or protocol is chosen. It computes nothing on secrets. Its output is public, the same for every party, and marks every step as local, as a multiplication with a triple, or as an opening. Appendix B gives this list for E_K step by step; counting its multiplications gives the 147 of §2.3. Every party then runs that same list under a chosen protocol, on its own shares. Because the list is fixed first, the number of multiplications, and with it the cost accounting of §2.3, is a property of Hydra and of the derivation, not of the protocol that later runs it.

Remark 7.1 (Protocol independence). The protocol enters only in which parties connect, in how preprocessing is produced, and in how a value is opened. We ran the same instruction list under three protocols: MASCOT, with $N = 2$, designed for security against up to $N - 1$ actively corrupted parties, and Shamir sharing with $N = 5$, $T = 2$, semi-honest and malicious, each designed to tolerate up to T corrupted parties. All gave identical outputs on identical inputs,

for the PRF alone and for the full derivation. Under every protocol, all N parties must take part in every run: the threshold T bounds how many parties may be corrupted, not how many may be offline.

7.2 Setup

Both implementations compute one complete hardened derivation on shares, from shared parent key and chain code to shared child key and chain code, including the BIP32 rule $k_{child} = (I_L + k_{par}) \bmod n$ and the opened validity verdict.

- *Hydra:* the derivation program of §7.1, with arithmetic shares over $\mathbb{F}_p$, i.e. Algorithms 1 and 3.
- *HMAC-SHA512:* `hmac_derive.mpc`, with binary shares. SHA-512 is evaluated with the standard Bristol Fashion circuit for its compression function [4] (57,947 AND gates). With a 32-byte key and a 37-byte message, HMAC-SHA512 is exactly four compressions. Two of them depend only on the chain code c_{par} (the inner and outer padded key blocks), so a wallet can compute them once per parent and reuse them for every child. We report this per-parent part separately; the other two compressions and the addition modulo n are per child. The addition, the comparisons with n and the zero test are binary circuits on the same shares. The implementation was checked against a standard HMAC-SHA512 library on random parents.

We use MP-SPDZ 0.2.8 [22], the version with which the Hydra designers benchmarked their design and which they distribute with their reference code [20, §8 and Appendix A]; we did not check whether later releases produce random bits under malicious Shamir sharing in the same way. We use $N = 5$ parties and threshold $T = 2$ unless stated otherwise. Following Das et al. [13, §6], the comparison is made under *malicious Shamir sharing* (`malicious-shamir-party.x`), which aims at security with abort against up to T parties that deviate from the protocol. It uses the method of Lindell and Nof [26]: triples are produced with the semi-honest protocol, checked by sacrificing, and used for Beaver multiplication, and every opening is checked. Random bits are produced differently, and Remark 6.15 explains why our proofs do not cover this implementation as it stands. For reference, the tables also give Hydra under *semi-honest* Shamir sharing (`shamir-party.x`); §7.7 discusses the difference, including the semi-honest figures for HMAC-SHA512. Both programs run unchanged under both protocols, and in both the same binary handles arithmetic and binary shares, so each comparison differs only in the derivation function. We checked the outputs of both programs against the reference implementations under both protocols. All parties run on one laptop (Intel Core i5-12450H; Docker with 4 cores and 8 GB of memory). Network round-trip time (RTT) and bandwidth are emulated with Linux `tc netem` on the loopback interface. Each configuration was run 3 to 20 times, except for a single run of HMAC-SHA512 at 25 ms; we report medians. Rounds, data and multiplication counts were identical in every run of every configuration; only the times vary. Under added delay the times of repeated runs differ by at most 14% of their median; without delay, where most runs take well under a second, by up to 38%.

Time is MP-SPDZ's measurement of the online computation, excluding compilation and process start-up. *Rounds* and *data* are MP-SPDZ's counts for party P_1 (MP-SPDZ's party 0), whose output we recorded; MP-SPDZ also reports the data sent by all parties together, which we give where it matters. *CPU time* and *peak memory* are measured per party process with GNU `time`. CPU cycles are estimated as CPU time times the processor's clock range (2.0–4.4 GHz), since hardware counters are not available in this environment.

7.3 One derivation

Table 2 gives the cost of one derivation without network delay. Under malicious Shamir sharing, HYDRA needs 63× fewer communication rounds than HMAC-SHA512 (243 against 15,226, by the round counter of the measured implementation). It is 10× faster per derivation, and 5× faster than the per-child part of HMAC-SHA512 alone; it needs 8.5× less CPU time per party and sends 2.4× less data.

Table 2: One hardened derivation, $N = 5$, $T = 2$, no network delay. The first two columns use malicious Shamir sharing (medians of 10 and 20 runs), and the ratio compares them; the last column gives HYDRA under semi-honest Shamir sharing for reference (median of 20 runs). Rounds and data are those of party P_1 unless stated otherwise; CPU time and memory are per party.

	Malicious Shamir			Semi-honest
	HMAC-SHA512	HYDRA	Ratio	HYDRA
Time per derivation	3.32 s	0.327 s	10×	0.155 s
per child only (pads reused)	1.75 s	0.327 s	5.3×	0.155 s
Fastest and slowest run	3.21–4.08 s	0.316–0.361 s		0.145–0.167 s
Communication rounds	15,226	243	63×	351
Data sent	53.4 MB	22.1 MB	2.4×	6.75 MB
Data sent, all parties	267 MB	108.6 MB	2.5×	31.7 MB
CPU time	2.07 s	0.245 s	8.5×	0.13 s
CPU cycles (estimated)	4.1–$9.1 \cdot 10^9$	0.5–$1.1 \cdot 10^9$		0.3–$0.6 \cdot 10^9$
Peak memory	32 MB	39 MB		29 MB
Multiplications	234,348 in $\mathbb{F}_2$	10,120 in $\mathbb{F}_p$		10,120 in $\mathbb{F}_p$
Derivations per minute (sequential)	≈ 18	≈ 180		≈ 390
Compilation, once per program	19.8 s	3.4 s		3.4 s

Table 4 in Appendix E splits each derivation into its parts, measured with timers inside the programs. Without delay, HYDRA's time under malicious Shamir sharing is dominated by preprocessing. MP-SPDZ produces checked triples and random bits in batches when they are first needed, and the timers charge each batch to the part that requests it: the first batches fall into the bit decomposition of k_{par}, which the program runs first (45%), and a second batch of triples into the addition modulo n. The split without delay therefore reflects the order of the program, not the cost of the parts. Under latency, the PRF dominates (51% at 100 ms), since it has the longest chain of dependent multiplications. Under semi-honest Shamir sharing the PRF takes 53% without delay and 71% at 100 ms. For HMAC-SHA512 the binary addition modulo n is negligible; nearly all of the cost is the four SHA-512 compressions.

7.4 Network latency

Latency adds to the cost of every communication step that has to wait for the previous one. MP-SPDZ's round counter also counts message batches that are sent in parallel, so it is higher than the number of sequential round trips. Table 3 shows the time for increasing RTT. Under malicious Shamir sharing, a least-squares fit over all measured RTT values gives slopes of 0.138 s per millisecond of RTT for HYDRA (0 to 100 ms) and 8.32 s per millisecond for HMAC-SHA512 (0 to 25 ms). For both the fit is within about 5% of the measurements for RTT $\geq$ 5 ms and overestimates below that, where local computation overlaps with the delay. The slopes mean about 140 and 8,300 sequential round trips per derivation. Their ratio, 60, is close to the ratio of the round counters, $15{,}226/243 = 63$, although each counter is higher than the number of sequential round trips. At 10 ms RTT, a typical value within one region, HYDRA takes 1.9 s and HMAC-SHA512 91 s, of which 1.6 s and 88 s are spent waiting for the network. At 100 ms RTT,

a typical intercontinental value, HYDRA takes 14 s while HMAC-SHA512 would take about 14 minutes. From 5 ms of RTT on, HYDRA is faster under malicious than under semi-honest Shamir sharing, because it needs fewer rounds there (243 against 351); §7.7 explains why.

Table 3: Time of one derivation versus network round-trip time, $N = 5$, $T = 2$. The first two columns use malicious Shamir sharing (medians of 5 runs; 10 and 20 without delay; HMAC-SHA512 at 25 ms is a single run); the last gives HYDRA under semi-honest Shamir sharing for reference. HMAC-SHA512 at 50 and 100 ms is extrapolated from the linear fit (5.2 s + 8.32 s per ms of RTT), as single runs would take 7–14 minutes. Table 2 gives the fastest and slowest run without delay; with delay, every run lies within 6% of its median, except the slowest runs of HYDRA at 1 ms under semi-honest (8% above) and at 5 ms under malicious Shamir sharing (11% above).

	Malicious Shamir			Semi-honest
RTT	HMAC-SHA512	HYDRA	Ratio	HYDRA
0 ms	3.32 s	0.327 s	10×	0.155 s
1 ms	11.4 s	0.472 s	24×	0.35 s
5 ms	49.3 s	1.08 s	46×	1.23 s
10 ms	91.4 s	1.91 s	48×	2.33 s
25 ms	211 s	4.07 s	52×	5.38 s
50 ms	(≈ 421 s)	7.52 s	(≈ 56×)	9.99 s
100 ms	(≈ 837 s)	14.1 s	(≈ 59×)	19.3 s

7.5 Bandwidth and number of parties

Table 5 (Appendix E) limits the bandwidth. Under malicious Shamir sharing HYDRA sends less data than HMAC-SHA512, and it remains faster at every limit tested, by 3.4× at 10 and 100 Mbit/s. The limit applies to the loopback interface as a whole, so all five parties share it (108.6 MB in total for one HYDRA derivation and 267 MB for HMAC-SHA512, as reported by MP-SPDZ for all parties together). With a separate link between each pair of parties, as in a real deployment, all columns would be lower. Under semi-honest Shamir sharing HYDRA sends less data still (31.7 MB in total), which is why it is faster there under a bandwidth limit.

Table 6 (Appendix E) varies the number of parties N and the threshold T. The number of communication rounds barely depends on either (under malicious Shamir sharing 242 to 245 for HYDRA, growing slightly with T, and 15,226 throughout for HMAC-SHA512), so the latency results of §7.4 hold for every configuration. The data each party sends grows with both N and T for both mechanisms. HYDRA is 8–16× faster in every configuration. With more than four parties, all processes share four cores, so the times for $N \geq 5$ are inflated by the test machine; rounds and data are not affected.

7.6 A dishonest majority (MASCOT)

Malicious Shamir sharing needs an honest majority, $T < N/2$, as a threshold wallet with $T < N/2$ provides. To see the comparison when a majority may be corrupted, we ran both programs, unchanged, under MASCOT (`mascot-party.x`), which is designed to be secure with abort against up to $N - 1$ actively corrupted parties (Remark D.1). We use $N = 2$. HYDRA uses MASCOT's arithmetic shares over $\mathbb{F}_p$; HMAC-SHA512 uses MP-SPDZ's actively secure protocol for binary shares in the same binary. We checked both outputs against the reference implementations in this setting as well. Table 7 in Appendix E gives the results.

Without network delay, HMAC-SHA512 is faster here, by a factor of 1.7. The reason is visible in the per-part timers: HYDRA's PRF takes only 0.26 s, while the bit-level steps of §4.3 and §4.4 take 3.0 s, because shared random bits are expensive to produce with active security, more so

with MASCOT than with malicious Shamir sharing. HMAC-SHA512, by contrast, runs entirely on binary shares. With any network delay the round count decides, as under Shamir sharing: between 0 and 10 ms of round-trip time, Hydra's time grows by about 0.41 s per millisecond and that of HMAC-SHA512 by 17.3 s per millisecond on average, though not linearly (25.4 s already at 1 ms). Hydra is 6.9× faster at 1 ms and 24× faster at 10 ms. These runs use two parties, against five under Shamir sharing, so the settings are not directly comparable with each other.

7.7 Malicious against semi-honest Shamir sharing

Under semi-honest Shamir sharing, HMAC-SHA512 takes 2.59 s without delay, with 15,033 rounds and 0.94 MB of data for party P_1, and 92.0 s at a 10 ms round-trip time. Malicious security thus makes it slower without delay and makes it send far more data (53.4 MB), while its time under latency stays about the same. Hydra becomes about twice as slow without delay and sends about three times as much data, but it needs fewer rounds (243 against 351), so from 5 ms of RTT on it becomes faster (1.91 s against 2.33 s at 10 ms). Under semi-honest Shamir sharing Hydra sends about 7× more data than HMAC-SHA512; under malicious Shamir sharing it sends 2.4× less. Both figures are for MP-SPDZ's default batch size (§7.8).

MP-SPDZ's verbose statistics, which count rounds per type of message, explain why the malicious protocol needs fewer rounds (Table 8 in Appendix E). Under semi-honest Shamir sharing Hydra multiplies in two different ways. The PRF's multiplications are written as explicit Beaver steps (§6.4) and consume 147 preprocessed triples, plus one for the mask ρ of the validity test; their openings take 135 rounds. Every other multiplication, mainly the bit-level steps of §4.3 and §4.4, uses the protocol's native multiplication, a local product followed by resharing, which takes a further 216 rounds, including the production of the preprocessed material. Under malicious Shamir sharing every multiplication is a Beaver step with a checked triple (10,120 triples), so the PRF's and the bit-level multiplications are the same kind of operation and their openings take 216 rounds together. The triples are produced in advance in a few large batches, which take only 17 rounds but carry 12.5 of the 22.1 MB that P_1 sends (MP-SPDZ's breakdown does not attribute a further 7.2 MB); 10 further broadcast rounds are used by the checks of the triples.

So from 5 ms of RTT on, security against malicious parties costs Hydra no time in our measurements, and making the headline comparison in the stronger setting does not disadvantage Hydra. Most of the extra data is key-independent preprocessing, which could be produced before a derivation is requested; our times include it. The protocol of §6.4 is unaffected, since it treats every multiplication as one Mult command, however the backend realizes it.

7.8 Discussion and limitations

Under malicious Shamir sharing, the setting of the comparison, Hydra needs far fewer rounds than HMAC-SHA512 (63× fewer by the round counter of the measured implementation), is faster in every configuration we measured, needs far less CPU time, and sends less data. The protocol matters for data: under semi-honest Shamir sharing Hydra sends more data than HMAC-SHA512, and under MASCOT without network delay HMAC-SHA512 is faster. Threshold wallets usually run across machines in different locations, where latency rather than bandwidth dominates, and there the gap is largest: 48× at 10 ms and an extrapolated 59× at 100 ms. The results come with the following limitations.

- All parties ran on one machine, with network conditions emulated; parallel processes compete for four cores.
- Active security was benchmarked with malicious Shamir sharing ($N = 5$, $T = 2$) and with MASCOT ($N = 2$, §7.6). Security against malicious parties is proved with abort for any

protocol that realizes $\mathcal{F}_{\mathsf{ABB}}^{\text{abort}}$; we do not show that MP-SPDZ's malicious Shamir protocol is one, since its random bits may allow a selective failure (Remark 6.15).

- Under semi-honest Shamir sharing, MP-SPDZ produces the preprocessing material (triples, square pairs, random bits) on demand, in batches interleaved with the online phase, so the reported times include it and there is no separate offline time to report; the multiplication counts are the compiler's.
- MP-SPDZ produces preprocessing in batches of 10,000 by default. One Hydra derivation then produces 20,000 triples under malicious Shamir sharing and uses 10,120, and under semi-honest Shamir sharing it produces 10,000 triples and uses 148. Times and data include the unused material; we did not tune the batch size.
- Each run performs one derivation attempt. Retries (probability about $2^{-117.4}$) and batching several derivations into one run were not measured.
- CPU cycles are estimated from CPU time, not counted.
- The HMAC-SHA512 baseline is evaluated with a round count that grows with the circuit's depth. A garbled-circuit (BMR) evaluation has a constant number of rounds, which matters because our headline result is the round count. We did not measure such a baseline. Das et al. report that BMR within MP-SPDZ improved neither the runtime nor the communication of their SHA-512 baseline [13, §6].

8 Conclusion

Hardened BIP32 derivation in a threshold wallet does not need SHA-512 inside MPC. Replacing HMAC-SHA512 by the MPC-friendly PRF Hydra, and keeping everything after the serialized PRF output as BIP32 defines it, gives a derivation on secret shares that, under malicious Shamir sharing with five parties, needs 63× fewer communication rounds than HMAC-SHA512 in the same setting and is 48× faster at 10 ms round-trip time (§7). Every hardened child stays secret-shared and can derive further. The protocol is UC-secure for sequential callers (§6.1) against static, semi-honest adversaries, and with abort against static malicious adversaries, with no assumption on the PRF (§6); the malicious result does not cover MP-SPDZ's malicious Shamir protocol, which we measured (Remark 6.15). The derivation function provides parent–child hiding, sibling independence and hardened-child unlinkability under the offset-PRF assumption (§5), which is not implied by the published Hydra claim and is the main open point of this work.

Several directions remain open.

- The variant of Remark 5.25, which needs only plain PRF security, at the cost of departing from BIP32's node state.
- A proof that the protocols cited in Remark 6.15 realize $\mathcal{F}_{\mathsf{ABB}}^{\text{abort}}$; an implementation of malicious Shamir sharing that matches them, in particular in how it produces random bits, and its benchmarks; a test of the possible selective failure in MP-SPDZ's random bits; and the corresponding statements for MASCOT (Remark D.1).
- A constant-round (garbled-circuit) baseline for HMAC-SHA512 (§7.8).
- Master-key generation and threshold public-key computation, which are modelled ideally here (§6.8), and benchmarks across real networks rather than an emulated one.

Acknowledgments

We used the AI models Claude Opus 5.5 and Claude Fable 5.2 (Anthropic) in three ways: to edit the text of this paper, to help write the code of the implementation and of the test and benchmark scripts, and to check the proofs and the citations. The code was tested against the reference implementation of the Hydra designers and against an independent cleartext implementation (§7), and the authors checked every proof, citation and result. The authors take full responsibility for the content of this paper.

References

[1] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219. Springer, 2016. doi:10.1007/978-3-662-53887-6_7. Cryptology ePrint Archive, Paper 2016/492, https://eprint.iacr.org/2016/492.

[2] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. Feistel structures for MPC, and more. In *Computer Security – ESORICS 2019, Part II*, volume 11736 of *Lecture Notes in Computer Science*, pages 151–171. Springer, 2019. doi:10.1007/978-3-030-29962-0_8. Cryptology ePrint Archive, Paper 2019/397, https://eprint.iacr.org/2019/397.

[3] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Transactions on Symmetric Cryptology*, 2020(3):1–45, 2020. doi:10.13154/tosc.v2020.i3.1-45. Cryptology ePrint Archive, Paper 2019/426, https://eprint.iacr.org/2019/426.

[4] David Archer, Victor Arribas Abril, Steve Lu, Pieter Maene, Nele Mertens, Danilo Sijacic, and Nigel Smart. 'Bristol Fashion' MPC circuits. Circuit collection; the SHA-512 circuit used here is the copy distributed with MP-SPDZ, https://github.com/mkskeller/bristol-fashion, n.d. URL https://nigelsmart.github.io/MPC-Circuits/. Web pages, accessed 12 November 2026.

[5] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, 2017. doi:10.1007/s00145-015-9214-4. Cryptology ePrint Archive, Paper 2011/136, https://eprint.iacr.org/2011/136.

[6] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1992. doi:10.1007/3-540-46766-1_34.

[7] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC 1988)*, pages 1–10. ACM, 1988. doi:10.1145/62212.62213.

[8] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd IEEE Symposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145. IEEE, 2001. doi:10.1109/SFCS.2001.959888. Cryptology ePrint Archive, Paper 2000/067, https://eprint.iacr.org/2000/067.

[9] Certicom Research. SEC 2: Recommended elliptic curve domain parameters, version 2.0. Standards for Efficient Cryptography, January 2010. URL https://www.secg.org/sec2-v2.pdf.

[10] ChihYun Chuang, IHung Hsu, and TingFang Lee. A two-party hierarchical deterministic wallets in practice. In *Proceedings of the 20th International Conference on Security and Cryptography (SECRYPT 2023)*, pages 850–856. SciTePress, 2023. doi:10.5220/0012138700003555. Extended version: Cryptology ePrint Archive, Paper 2023/714, https://eprint.iacr.org/2023/714.

[11] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography (TCC 2006)*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006. doi:10.1007/11681878_15.

[12] Poulami Das, Andreas Erwig, Sebastian Faust, Julian Loss, and Siavash Riahi. The exact security of BIP32 wallets. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*, pages 1020–1042. ACM, 2021. doi:10.1145/3460120.3484807.

[13] Poulami Das, Andreas Erwig, Sebastian Faust, Philipp-Florens Lehwalder, Julian Loss, Ziyan Qu, and Siavash Riahi. BIP32-compatible threshold wallets. In *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security (ASIA CCS 2025)*, pages 856–872. ACM, 2025. doi:10.1145/3708821.3710830. Full version: Cryptology ePrint Archive, Paper 2023/312, https://eprint.iacr.org/2023/312.

[14] Christoph Dobraunig, Lorenzo Grassi, Anna Guinet, and Daniël Kuijsters. Ciminion: Symmetric encryption based on Toffoli-gates over large finite fields. In *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 3–34. Springer, 2021. doi:10.1007/978-3-030-77886-6_1. Cryptology ePrint Archive, Paper 2021/267, https://eprint.iacr.org/2021/267.

[15] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 823–852. Springer, 2020. doi:10.1007/978-3-030-56880-1_29. Cryptology ePrint Archive, Paper 2020/338, https://eprint.iacr.org/2020/338.

[16] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained PRFs. In *Advances in Cryptology – ASIACRYPT 2014, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 82–101. Springer, 2014. doi:10.1007/978-3-662-45608-8_5. Cryptology ePrint Archive, Paper 2014/416, https://eprint.iacr.org/2014/416.

[17] David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 88–102. IEEE, 2021. doi:10.1109/EuroSP51992.2021.00017. Cryptology ePrint Archive, Paper 2020/096, https://eprint.iacr.org/2020/096.

[18] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986. doi:10.1145/6490.6503.

[19] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a generalization of substitution-permutation networks: The HADES design strategy. In *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106

of *Lecture Notes in Computer Science*, pages 674–704. Springer, 2020. doi:10.1007/978-3-030-45724-2_23. Cryptology ePrint Archive, Paper 2019/1107, https://eprint.iacr.org/2019/1107.

[20] Lorenzo Grassi, Morten Øygarden, Markus Schofnegger, and Roman Walch. From Farfalle to Megafono via Ciminion: The PRF Hydra for MPC applications. In *Advances in Cryptology – EUROCRYPT 2023, Part IV*, volume 14007 of *Lecture Notes in Computer Science*, pages 255–286. Springer, 2023. doi:10.1007/978-3-031-30634-1_9. Full version: Cryptology ePrint Archive, Paper 2022/342, https://eprint.iacr.org/2022/342.

[21] Jens Groth and Victor Shoup. Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Paper 2022/506, 2022. URL https://eprint.iacr.org/2022/506.

[22] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS 2020)*, pages 1575–1590. ACM, 2020. doi:10.1145/3372297.3417872. Cryptology ePrint Archive, Paper 2020/521, https://eprint.iacr.org/2020/521.

[23] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS 2016)*, pages 830–842. ACM, 2016. doi:10.1145/2976749.2978357. Cryptology ePrint Archive, Paper 2016/505, https://eprint.iacr.org/2016/505.

[24] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-hashing for message authentication. RFC 2104, February 1997. URL https://www.rfc-editor.org/rfc/rfc2104.

[25] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC 2006)*, pages 109–118. ACM, 2006. doi:10.1145/1132516.1132532. Full version: SIAM Journal on Computing 39(5), pp. 2090–2112, 2010.

[26] Yehuda Lindell and Ariel Nof. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 2017)*, pages 259–276. ACM, 2017. doi:10.1145/3133956.3133999. Full version: Cryptology ePrint Archive, Paper 2017/816, https://eprint.iacr.org/2017/816.

[27] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979. doi:10.1145/359168.359176.

[28] Matthias Johann Steiner. Gröbner basis cryptanalysis of Ciminion and Hydra. *IACR Transactions on Symmetric Cryptology*, 2025(1):240–275, 2025. doi:10.46586/tosc.v2025.i1.240-275. Also arXiv:2405.05040.

[29] Pieter Wuille. BIP32: Hierarchical deterministic wallets. Bitcoin Improvement Proposal 32, 2012. URL https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki.

A Hydra instance and specification

This appendix presents the Hydra PRF [20] as specified by its authors, and our instance of it: the prime and the matrices M_I, M_J are ours, chosen to meet the conditions of the specification. Where the paper leaves a detail open, we follow the authors' reference implementation

`hydra.sage` [20][1]; our cleartext implementation reproduces that reference byte for byte at the authors' own parameters, including the round constants. HYDRA is an arithmetization-oriented pseudorandom function over $\mathbb{F}_p$, following the MEGAFONO design: a single keyed permutation (the *body*) is applied to the input, its internal states are summed to expand the state "for free," and a bank of keyed permutations (the *heads*) turns the result into pseudorandom output. All components use the prime, matrices, and nonlinear layers fixed in §A.1–§A.3. Remark A.5, at the end of this appendix, relates the specification to the rest of the paper.

A.1 Choice of prime

We require a prime $p > 2^{128}$, so that every 128-bit word x satisfies $0 \le x < 2^{128} < p$. This gives a *lossless embedding*: the map from 128-bit strings to $\mathbb{F}_p$ is injective and needs no modular reduction. HYDRA's external S-box layer S_E applies the power map $x \mapsto x^\alpha$, which is a permutation of $\mathbb{F}_p$ only when $\gcd(\alpha, p-1) = 1$, and for efficiency we want $\alpha = 3$.

The naive candidate $p = 2^{128} + 51$ fails this requirement: while it is prime, $\gcd(3,\, p-1) = 3$, so the cubing map is 3-to-1 and not a permutation. We select instead:

$$\boxed{p \;=\; 2^{128} + 385}$$

We verified that this value is prime and that $\gcd(3,\, p-1) = 1$, so $\alpha = 3$ is the smallest valid exponent. It is the smallest such prime: the primes $2^{128} + k$ with $k \le 385$ are those with $k \in \{51, 81, 165, 273, 385\}$, and only $k = 385$ has $\gcd(3, p-1) = 1$. The prime satisfies the requirements of the HYDRA specification: $p > 2^{63}$ and p odd [20, §1.5, §5.1], and the security level must satisfy $2^{80} \le 2^\kappa \le \min\{p^2, 2^{256}\}$, which admits our choice $\kappa = 128$. (The bound $\min\{2\log_2 p, 256\} = 256$ is the largest admissible κ at this prime, not the level of this instance; the round numbers of §2.4 are those for $\kappa = 128$.)

A.2 Matrices

The three matrices below are instantiated for $p = 2^{128} + 385$ and verified to satisfy all conditions required by the HYDRA specification. Verification was performed in SageMath using the procedure described in [20, Appendix H.1, Algorithm 8].

The external-round matrix $M_E \in \mathbb{F}_p^{4\times 4}$ is the standard MDS circulant matrix $\mathrm{circ}(3, 2, 1, 1)$ recommended by the HYDRA authors. It is MDS over $\mathbb{F}_p$ for our prime: all its square minors are nonzero modulo p.

$$M_E = \begin{pmatrix} 3 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 \\ 1 & 1 & 3 & 2 \\ 2 & 1 & 1 & 3 \end{pmatrix}$$

The internal-round matrix $M_I \in \mathbb{F}_p^{4\times 4}$ follows the form suggested in [20, Eq. (8)] and satisfies: (a) degree growth through the Lai–Massey construction, (b) density of the interpolation polynomial, and (c) resistance against infinitely long subspace trails (minimal polynomial of maximum degree and irreducible over $\mathbb{F}_p$, verified through Algorithm 8).

$$M_I = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 2 & 2 & 1 & 1 \\ 2 & 1 & 3 & 1 \\ 3 & 1 & 1 & 8 \end{pmatrix}$$

The head-round matrix $M_J \in \mathbb{F}_p^{8\times 8}$ satisfies analogous conditions to M_I over the 8-dimensional state space of the heads, with respect to the coefficient vector

[1] The file is in the repository named in [20, Appendix A], at the time of writing `https://extgit.isec.tugraz.at/krypto/hydra`.

$(1,1,1,1,-1,-1,-1,-1)$ used in S_J. It was found by randomized search over small-valued matrices of the form suggested in [20, §5.4], and verified via Algorithm 8.

$$M_J = \begin{pmatrix} 5 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 6 & 2 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 1 & 6 & 1 & 1 & 1 & 1 & 1 \\ 9 & 1 & 1 & 9 & 1 & 1 & 1 & 1 \\ 3 & 1 & 1 & 1 & 4 & 1 & 1 & 1 \\ 8 & 1 & 1 & 1 & 1 & 7 & 1 & 1 \\ 7 & 1 & 1 & 1 & 1 & 1 & 8 & 1 \\ 2 & 1 & 1 & 1 & 1 & 1 & 1 & 5 \end{pmatrix}$$

A.3 Nonlinear layers

HYDRA has three nonlinear layers, all designed to need few multiplications in MPC; each definition below gives its cost. Remark A.4 notes which of their operations need communication.

Definition A.1 (S_E: External Power Map)**.** The full power map $S_E : \mathbb{F}_p^4 \to \mathbb{F}_p^4$ is defined by

$$S_E(x_0, x_1, x_2, x_3) = (x_0^3,\ x_1^3,\ x_2^3,\ x_3^3).$$

MPC cost: 4 Beaver triples and 4 square pairs per round (one of each per word), multiplicative depth 1.

Definition A.2 (S_I: Internal Lai–Massey)**.** The generalized Lai–Massey map $S_I : \mathbb{F}_p^4 \to \mathbb{F}_p^4$ of algebraic degree 4 is defined by

$$y_l \;=\; x_l \;+\; \left(\left(\sum_{j=0}^{3}(-1)^j\, x_j\right)^2 \;+\; \left(\sum_{j=0}^{3}(-1)^{\lfloor j/2\rfloor}\, x_j\right)\right)^2, \qquad 0 \le l \le 3.$$

MPC cost: 2 Beaver triples per round (one per squaring), multiplicative depth 2, because the two squarings are sequential, the second taking the first one's output as part of its input. The two coefficient vectors $(1,-1,1,-1)$ and $(1,1,-1,-1)$ are linearly independent and their entries sum to zero, so the map is invertible by [20, Proposition 1].

Definition A.3 (S_J: Head Lai–Massey)**.** The generalized Lai–Massey map $S_J : \mathbb{F}_p^8 \to \mathbb{F}_p^8$ of algebraic degree 2 is defined by

$$y_l \;=\; x_l \;+\; \left(\sum_{h=0}^{7}(-1)^{\lfloor h/4\rfloor}\, x_h\right)^2, \qquad 0 \le l \le 7.$$

MPC cost: 1 Beaver triple per round, depth 1.

Remark A.4. All linear combinations in S_I and S_J are evaluated locally on shares (zero communication cost). Only the squarings consume Beaver triples.

A.4 Interface and constants

HYDRA is keyed by $K \in \mathbb{F}_p^4$ and maps an input $m = [\nu \,\|\, IV] \in \mathbb{F}_p^4$, consisting of a nonce $\nu \in \mathbb{F}_p$ and a fixed initial value $IV \in \mathbb{F}_p^3$, to an output of at least four words. Its security is claimed for an attacker whose data is limited to $2^{\kappa/2}$ outputs per key [20, §5.1]. Each head emits 8 words, and further heads extend the output in blocks of 8. We use four output words: a single head, whose output is truncated to its first four words, which suffices for a 512-bit output.

The round constants are generated as in the reference implementation: SHAKE-128 is seeded with the ASCII string `Hydra`, a label, and the prime p as little-endian 64-bit blocks, and its output

is read in $\lceil \log_2 p/8 \rceil$-byte chunks and rejection-sampled into $\mathbb{F}_p$. The label `Constants` yields the body constants (in order: the 2 initial external, the R_I internal, and the 4 final external rounds) followed by the head constants. The matrices M_I, M_J of §A.2 replace the ones the reference derives from the label `Matrices`, which does not affect the constants because each label seeds its own stream.

A.5 Body

The body B is a Hades-style permutation of $\mathbb{F}_p^4$ with an asymmetric external split, combined with the MEGAFONO accumulation. On input $s^{(0)} \in \mathbb{F}_p^4$ it applies, in order, 2 initial external rounds, R_I internal rounds, and 4 final external rounds. An external round is

$$\mathcal{E}(s) \;=\; M_E \cdot S_E(s) \;+\; \gamma,$$

with S_E the full power map (Definition A.1) and γ the public round constant. An internal round is

$$\mathcal{I}(s) \;=\; M_I \cdot S_I(s) \;+\; \gamma,$$

with S_I the internal Lai–Massey map (Definition A.2). Writing $s^{(r)}$ for the state after round r and $R_E + R_I$ for the total number of body rounds, the MEGAFONO sum accumulates every intermediate state except the last:

$$z \;=\; \sum_{r=1}^{R_E+R_I-1} s^{(r)} \;\in\; \mathbb{F}_p^4, \qquad y \;=\; s^{(R_E+R_I)} \;\in\; \mathbb{F}_p^4.$$

The final round's state is excluded from the sum because it is already carried separately as y. This is how the reference implementation `hydra.sage` computes z, and our implementation matches its outputs exactly. The body returns the pair (y, z): the final state y and the accumulated sum z that provides the "free" state expansion.

A.6 Key expansion

The heads are keyed by the expanded key

$$K' \;=\; K \,\|\, (M_E \cdot K) \;\in\; \mathbb{F}_p^8,$$

obtained from the PRF key $K \in \mathbb{F}_p^4$ by a single linear map. No additional key material is introduced.

A.7 Heads

Each head is a keyed permutation $H_{K'}$ of $\mathbb{F}_p^8$ built from R_H Lai–Massey rounds with per-round key injection. A head round is

$$\mathcal{J}(v) \;=\; M_J \cdot S_J(v) \;+\; \gamma,$$

with S_J the head Lai–Massey map (Definition A.3), and the expanded key K' is added after each of the R_H rounds:

$$H_{K'}(v) \;=\; K' + \mathcal{J}_{R_H-1} \circ (K' + \mathcal{J}_{R_H-2}) \circ \cdots \circ (K' + \mathcal{J}_0)(v).$$

There is no key addition before the first round; the raw input enters $\mathcal{J}_0$ directly. Additional heads use the same $H_{K'}$ but a different input: head i is applied to $\mathcal{R}_i(y, z)$, where $\mathcal{R}_i$ is a nonlinear rolling function of the body output [20, §5.3]. With four output words only the first head is used and no rolling function is evaluated.

A.8 The PRF

HYDRA composes the above with Even–Mansour-style whitening and a feed-forward:

1. *Pre-whitening.* $s \leftarrow M_E \cdot (m + K)$.
2. *Body.* $(y_{\text{raw}}, z) \leftarrow B(s)$, then $y \leftarrow y_{\text{raw}} + K$.
3. *Head input.* $v \leftarrow y \,\|\, z \in \mathbb{F}_p^8$.
4. *Head with feed-forward.* $\text{out} \leftarrow H_{K'}(v) + v$.
5. *Truncation.* $E_K(m) \leftarrow (\text{out}_0, \text{out}_1, \text{out}_2, \text{out}_3) \in \mathbb{F}_p^4$ (the first four words of the 8-word head output).

Remark A.5. This is the plain HYDRA PRF over $\mathbb{F}_p$. Nothing in this appendix is specific to BIP32, to secret sharing, or to our derivation setting; those appear only in §4. The security of E_K as a PRF is the claim made in [20]. Section 5.8 states exactly what our construction needs beyond it (Assumption 5.21).

B Compiling Hydra for MP-SPDZ

Algorithm 2 is E_K of Appendix A as our PRF program (§7.1) hands it to the MP-SPDZ compiler, round by round. Every step is marked LOCAL (no communication), BEAVER (one multiplication with a triple) or OPEN (one public reconstruction). Counting the marks gives the 147 multiplications of §2.3. Nothing in the count depends on the number of parties, the threshold, or the protocol: only on p and on whether the square-pair shortcut for cubing is used.

At run time every party executes each BEAVER mark as one multiplication with a preprocessed triple, and each OPEN mark as an opening.

Algorithm 2 Compiling E_K (Appendix A) to a protocol-agnostic tape, as the PRF program emits it for a single output block.

Input: Prime p; flag UseSquares. The output length is fixed at four words, the single 4-word block of §A.8, so exactly one head permutation is emitted

Output: Instruction tape $\mathcal{T}$, every entry tagged Local (no communication), Beaver (one triple) or Open (one reconstruction, no triple); $\mathcal{T}$ is independent of the party count, the threshold and the protocol

procedure Compile(p, UseSquares)
  $R_{E1} \leftarrow 2$, $R_{E2} \leftarrow 4$, $R_I \leftarrow 42$, $R_H \leftarrow 39$; fix M_E, M_I, M_J ▷ *from* $(p, \kappa{=}128)$*, §2.4*
  $\mathcal{T} \leftarrow [$ Local : $s \leftarrow M_E \cdot (m + K)\ ]$; $z \leftarrow (0,0,0,0)$ ▷ *pre-whitening, §A.8 step 1*
  Body (§A.5): three positional groups: R_{E1} external, then R_I internal, then R_{E2} external
  for R_{E1} rounds **do**
    EmitCubes(s, UseSquares) ▷ *S_E, Definition A.1*
    append Local: $s \leftarrow M_E \cdot s + \gamma$; $z \leftarrow z + s$ ▷ *$\mathcal{E}(s)$, then Megafono accumulation*
  for R_I rounds **do**
    append Local: $u_1 \leftarrow s_0 - s_1 + s_2 - s_3$; $u_2 \leftarrow s_0 + s_1 - s_2 - s_3$ ▷ *the two Lai–Massey dots*
    append Beaver: $q \leftarrow u_1 \cdot u_1$
    append Beaver: $q_2 \leftarrow (q + u_2) \cdot (q + u_2)$ ▷ *Definition A.2: 2 triples, and they are sequential*
    append Local: $s_l \leftarrow s_l + q_2$ for $0 \le l \le 3$; $s \leftarrow M_I \cdot s + \gamma$; $z \leftarrow z + s$ ▷ *$\mathcal{I}(s)$, then accumulation*
  for $j \leftarrow 1$ to R_{E2} **do**
    EmitCubes(s, UseSquares)
    append Local: $s \leftarrow M_E \cdot s + \gamma$
    if $j < R_{E2}$ **then**
      append Local: $z \leftarrow z + s$ ▷ *the final body round is* not *accumulated*
  append Local: $y \leftarrow s + K$; $K' \leftarrow K \,\|\, M_E K$; $v \leftarrow y \,\|\, z$ ▷ *§A.8 steps 2–3; §A.6*
  Head (§A.7): R_H rounds on a working copy u, so the feed-forward can still read v
  $u \leftarrow v$ ▷ *v is never overwritten*
  for R_H rounds **do**
    append Local: $\mathrm{dot} \leftarrow u_0 + u_1 + u_2 + u_3 - u_4 - u_5 - u_6 - u_7$
    append Beaver: $g \leftarrow \mathrm{dot} \cdot \mathrm{dot}$ ▷ *Definition A.3: 1 triple*
    append Local: $u_l \leftarrow u_l + g$ for $0 \le l \le 7$; $u \leftarrow M_J \cdot u + \gamma + K'$ ▷ *$\mathcal{J}(u)$, then key injection*
  append Local: $\mathrm{out} \leftarrow u + v$ ▷ *feed-forward onto the saved v, §A.8 step 4*
  append Local: $E_K(m) \leftarrow (\mathrm{out}_0, \mathrm{out}_1, \mathrm{out}_2, \mathrm{out}_3)$ ▷ *truncation, §A.8 step 5*
  Totals per evaluation: with UseSquares, $4R_E + 2R_I + R_H = 147$ Beaver steps and $4R_E = 24$ square pairs; without it, $8R_E + 2R_I + R_H = 171$ and no square pairs
  return $\mathcal{T}$

procedure EmitCubes(s, UseSquares) ▷ *$s_l \leftarrow s_l^3$ for $0 \le l \le 3$*
  for $l \leftarrow 0$ to 3 **do**
    if UseSquares **then**
      append Local: take a fresh square pair (r, r^2) from preprocessing
      append Beaver: $r^3 \leftarrow r \cdot r^2$ ▷ *independent of s_l*
      append Open: $\eta \leftarrow s_l - r$ ▷ *the only opening that depends on s_l*
      append Local: $s_l \leftarrow 3\eta r^2 + 3\eta^2 r + \eta^3 + r^3$ ▷ *$= (\eta + r)^3 = s_l^3$; public coefficients, so free*
    else
      append Beaver: $q \leftarrow s_l \cdot s_l$; append Beaver: $s_l \leftarrow q \cdot s_l$ ▷ *2 triples, depth 2*

C Child scalar and limb arithmetic

This appendix gives the procedure behind the AddModN step and the validity block of Algorithm 1 (Algorithm 3), and the limb arithmetic it relies on (§C.1).

Algorithm 3 Child scalar and chain code, executed by party P_j.

Input: Limb shares $[\![\omega_0]\!]_j, [\![\omega_1]\!]_j$ $(= I_L)$ and $[\![\omega_2]\!]_j, [\![\omega_3]\!]_j$ $(= I_R)$; parent key limb shares $[\![k^{\mathrm{hi}}]\!]_j, [\![k^{\mathrm{lo}}]\!]_j$; public $n = (n^{\mathrm{hi}}, n^{\mathrm{lo}})$

Output: Child limb shares $[\![k_{child}]\!]_j, [\![c_{child}]\!]_j$; public verdict bad

procedure ChildScalar($[\![\omega_0]\!]_j, [\![\omega_1]\!]_j, [\![\omega_2]\!]_j, [\![\omega_3]\!]_j,\ [\![k^{\mathrm{hi}}]\!]_j, [\![k^{\mathrm{lo}}]\!]_j$)
  $[I_L]_j \leftarrow$ BitDecompose($[\![\omega_0]\!]_j, [\![\omega_1]\!]_j$) ▷ *256 shared bits; from the overflow test*
  $[k_{par}]_j \leftarrow$ BitDecompose($[\![k^{\mathrm{hi}}]\!]_j, [\![k^{\mathrm{lo}}]\!]_j$) ▷ *256 shared bits; once per level*
  AddModN: $k_{child} = (I_L + k_{par}) \bmod n$
  $[S]_j \leftarrow$ BitAdd($[I_L]_j,\ [k_{par}]_j$) ▷ *257 shared bits; $S < 2n$ if $I_L < n$*
  $[\![g]\!]_j \leftarrow$ SecureGE($[S]_j,\ n$) ▷ *shared bit, not opened*
  $([\![S^{\mathrm{hi}}]\!]_j, [\![S^{\mathrm{lo}}]\!]_j) \leftarrow$ BitRecompose($[S]_j$) ▷ *local*
  $[\![\beta]\!]_j \leftarrow [\![g]\!]_j \cdot (1 -$ SecureGE($[S^{\mathrm{lo}}]_j,\ n^{\mathrm{lo}}$)) ▷ *borrow of $S - n$*
  $[\![k^{\mathrm{lo}}_{child}]\!]_j \leftarrow [\![S^{\mathrm{lo}}]\!]_j - n^{\mathrm{lo}}[\![g]\!]_j + 2^{128}[\![\beta]\!]_j$ ▷ *local; CondSubN*
  $[\![k^{\mathrm{hi}}_{child}]\!]_j \leftarrow [\![S^{\mathrm{hi}}]\!]_j - n^{\mathrm{hi}}[\![g]\!]_j - [\![\beta]\!]_j$ ▷ *local*
  Validity (BIP32 rule)
  bad $\leftarrow$ SecureGE($[I_L]_j,\ n$) $\vee$ SecureIsZero($[\![k_{child}]\!]_j$) ▷ *on $[S]_j$, see below; open one bit*
  if bad $= 1$ **then return** $(\perp, \perp,\ \text{bad} = 1)$ ▷ *rejection*
  $[\![c_{child}]\!]_j \leftarrow ([\![\omega_2]\!]_j,\ [\![\omega_3]\!]_j)$ ▷ $c_{child} = I_R$
  return $(([\![k^{\mathrm{hi}}_{child}]\!]_j, [\![k^{\mathrm{lo}}_{child}]\!]_j),\ [\![c_{child}]\!]_j,\ \text{bad} = 0)$

C.1 Limb arithmetic on $\mathbb{F}_p$ shares

BitAdd is a carry-lookahead adder on the two 256-bit vectors; its 257 output bits are the sum $S = I_L + k_{par}$. On attempts with $I_L < n$ we have $S < 2n$, and the rest of this subsection is about these attempts; attempts with $I_L \geq n$ are rejected by the validity test whatever their candidate limbs are (we tested both kinds of attempt at their boundaries, §7.1). SecureGE against the public n gives the shared bit $g = [S \geq n]$, which is not opened. The conditional subtraction $k_{child} = S - g\,n$ is then done on limbs, which is safe here because every true result lies in $[0, 2^{128}) \subset \mathbb{F}_p$. The parties recompose S^{lo} from bits 0–127 and S^{hi} from bits 128–256 (local), and compute the borrow $\beta = g \cdot [S^{\mathrm{lo}} < n^{\mathrm{lo}}]$ with one further 128-bit comparison. Then

$$k^{\mathrm{lo}}_{child} = S^{\mathrm{lo}} - g\,n^{\mathrm{lo}} + 2^{128}\beta, \qquad k^{\mathrm{hi}}_{child} = S^{\mathrm{hi}} - g\,n^{\mathrm{hi}} - \beta$$

are local. Both left-hand sides are the true limbs of $S - g\,n \in [0, n)$, so their field values are exact, even though S^{hi} itself can exceed p.

Since $S < 2n$ on these attempts, $k_{child} = 0$ exactly when $S = 0$ or $S = n$; the first cannot occur, since $k_{par} \geq 1$, but testing it costs nothing extra. Both are linear in the bits of S: $S = 0$ iff its popcount is 0, and $S = n$ iff its Hamming distance from the public bits of n is 0. Each count lies in $[0, 257]$, so their product is a field element that is zero exactly when $k_{child} = 0$, at the cost of one multiplication.

BitDecompose can be realized with edaBits [15], or, as in our implementation, from individually generated shared random bits. The implementation opens both parts of the verdict together: it forms the product $(1 - [I_L \geq n]) \cdot \mathrm{popcount}(S) \cdot \mathrm{dist}(S, n)$, multiplies it by a fresh shared random value and opens the result (Definition 6.5). It reuses the decomposition of ω_0, ω_1

that the overflow test of §4.3 computes anyway, so I_L is not decomposed twice, and it decomposes k_{par} once per level, not once per attempt.

D Full UC proof

D.1 The simulator

In the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model the view of a corrupted party consists of the public commands, the public program, the values sent by Open and PubKey, and the values sent to it by OpenTo. The masked operands of Beaver multiplications and the messages of bit decompositions are not part of it at this level: they belong to the realization of $\mathcal{F}_{\mathsf{ABB}}$, and are handled in Corollary 6.8. The simulator runs $\mathcal{A}$ internally. For every command, it learns the command's name and public arguments from $\mathcal{F}_{\mathsf{HD}}$, which is what lets it produce the public sequence of $\mathcal{F}_{\mathsf{ABB}}$ commands even when no party is corrupted, and it produces the opened values as follows.

- Init: nothing is opened, so nothing is simulated.
- Derive(w, i): receive $\lambda(w, i)$ from $\mathcal{F}_{\mathsf{HD}}$ and follow the public program attempt by attempt, using λ to decide the control flow. In each attempt:
  1. for each of the $4R_E = 24$ cube openings $s_l - r$, output a fresh uniform element of $\mathbb{F}_p$;
  2. for the overflow opening, output the next bit ovf of λ;
  3. if ovf $= 0$, for the validity opening, output 0 if the next bit bad of λ is 1, and a fresh uniform element of $\mathbb{F}_p \setminus \{0\}$ otherwise.
- Open(w): output the values (k_w, c_w) received from $\mathcal{F}_{\mathsf{HD}}$, split into limbs.
- Export(w, j): if P_j is honest, nothing is opened to the adversary, so nothing is simulated; if P_j is corrupted, output the values (k_w, c_w) received from $\mathcal{F}_{\mathsf{HD}}$, split into limbs, as P_j's received values.
- Pub(w): output the point P_w received from $\mathcal{F}_{\mathsf{HD}}$.

Finally $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. $\mathcal{S}$ uses no secret and runs in time linear in that of $\mathcal{A}$ and the length of the program.

D.2 Proof of the security theorem

We first prove the correctness lemma, Lemma 6.6, and then Theorem 6.7 with the simulator of §D.1.

Proof of Lemma 6.6. By induction over the nodes in the order of creation; the root holds by RootGen. For one attempt at node w:

- The PRF part is a straight-line program of Lin and Mult that computes E_K exactly (Appendix B); the cube step is exact because $3\eta r^2 + 3\eta^2 r + \eta^3 + r^3 = (\eta + r)^3 = s_l^3$.
- Every output word satisfies $\omega_l < p < 2^{129}$, so bit 128 of its 129-bit decomposition is 1 exactly when $\omega_l \geq 2^{128}$; hence ovf is the overflow verdict.
- On an attempt with $I_L < n$, the addition modulo n is exact by §C.1: the adder works on bits, and the conditional subtraction produces limbs whose true values lie in $[0, 2^{128}) \subset \mathbb{F}_p$. On an attempt with $I_L \geq n$ the candidate limbs need not be canonical; they are not used, because the attempt is rejected (next item).

- popcount(S) and dist(S, n) lie in $[0, 257]$, so their product is at most $257^2 < p$ and is zero in $\mathbb{F}_p$ exactly when one of them is zero. Hence $v = 0$ exactly when $I_L \geq n$ or $k_{child} = 0$ (§C.1), and, since the attempt is not faulty, $y = \rho v = 0$ exactly when $v = 0$. So bad is the validity verdict.

The verdicts, and with them the control flow and the accepted counter, are therefore those of Algorithm 1 run in the clear, and the stored child limbs are $\mathsf{Tab}[w|\sigma(w, i)]$. □

Proof of Theorem 6.7. We go from the real execution to the ideal one through four hybrids, for a fixed $\mathcal{A}$ and $\mathcal{Z}$. Each hybrid is the joint distribution of the corrupted parties' view and the honest parties' outputs.

H_0 is the real execution of Π_{HD} in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model.

H_1 is H_0 with each ρ drawn uniformly from $\mathbb{F}_p \setminus \{0\}$ instead of $\mathbb{F}_p$. In H_1 no attempt is faulty, so Lemma 6.6 applies to every node. The number of attempts is random, and in H_0 a mask $\rho = 0$ can cause an extra attempt, so we bound $\mathrm{SD}(H_0, H_1)$ by a coupling. For every attempt that reaches the validity test, in the order in which they occur, sample an independent $\rho' \leftarrow \mathbb{F}_p \setminus \{0\}$, used as the mask in H_1, and an independent coin that shows 1 with probability $1/p$; H_0 uses the mask 0 if the coin shows 1 and ρ' otherwise, so that each mask of H_0 is uniform on $\mathbb{F}_p$. Let Q be the number of attempts that reach the validity test in H_1. It depends only on the randomness of H_1, so it is independent of the coins, and if none of the first Q coins shows 1, the two executions are identical and end after the same Q attempts. Hence

$$\mathrm{SD}(H_0, H_1) \;\leq\; \Pr[\text{one of the first } Q \text{ coins shows 1}] \;\leq\; \frac{\mathbb{E}[Q]}{p} \;\leq\; \frac{q_{\mathrm{att}}}{p},$$

where the last step holds because H_1, H_2 and H_3 are identically distributed (below) and H_3 is the ideal execution, whose number of such attempts q_{att} bounds.

H_2 is H_1 with each opened y of an accepted validity test replaced by a fresh uniform element of $\mathbb{F}_p \setminus \{0\}$. When $v \neq 0$, the map $\rho \mapsto \rho v$ is a bijection of $\mathbb{F}_p \setminus \{0\}$, and ρ is used nowhere else, so y is already uniform on $\mathbb{F}_p \setminus \{0\}$ and independent of everything else. When $v = 0$, $y = 0$ in both. Hence $H_2 = H_1$.

H_3 is H_2 with each opened cube value $\eta = s_l - r$ replaced by a fresh uniform element of $\mathbb{F}_p$. Each r is a fresh output of Rand, and is used only to form r^2, r^3 and the stored value $s_l^3 = (\eta + r)^3$, none of which is opened. Every other opened value is a function of the derivation values and of the ρ's alone, because s_l^3 does not depend on r. Conditioned on all of these, η is therefore uniform and independent of everything else, jointly over all openings since each uses its own r. Hence $H_3 = H_2$.

H_3 is the ideal execution. By Lemma 6.6, in H_3 the opened verdict bits are $\lambda(w, i)$, the accepted counters are $\sigma(w, i)$, the nodes created and their stored values are those of Tab, the values opened by Open(w) and Export(w, j) are $\mathsf{Tab}[w]$, and the point returned by PubKey in Pub(w) is $k_w \cdot G$. The remaining opened values are fresh uniform elements of $\mathbb{F}_p$, or of $\mathbb{F}_p \setminus \{0\}$, placed as the program dictates. This is exactly the view $\mathcal{S}$ produces from λ and the Open, Export and Pub outputs that $\mathcal{F}_{\mathsf{HD}}$ gives the adversary, and the honest parties' outputs (init), $(\mathrm{derived}, w, i, \sigma(w, i))$, (k_w, c_w) and P_w are exactly those of $\mathcal{F}_{\mathsf{HD}}$.

Summing, $\mathcal{Z}$ distinguishes the real and ideal executions with advantage at most q_{att}/p. □

D.3 Proofs of the corollaries

This subsection proves the two corollaries of §6.6: the Shamir instantiation (Corollary 6.8) and the wallet guarantees (Corollary 6.9).

Proof of Corollary 6.8. The realizations of Lin, Mult, Rand, Open and OpenTo are the standard semi-honest Shamir protocols. Their security against at most $T < N/2$ semi-honest

parties is perfect, with straight-line simulators, which suffices for security under universal composition [25]; see [5] for a complete treatment of BGW-style Shamir protocols. In a Beaver step the two opened values $x - a$ and $y - b$ are uniform and independent of x, y, because a, b are uniform and used once, so that step is perfect as well.

The bit decomposition is only statistically secure, so we do not rely on the perfect-security composition result for it and simulate it directly. We first state the algorithm, which is MP-SPDZ's `BitDecFull` as compiled for our prime. Let $a \in [0, p)$ be the value to decompose.

1. Take 128 fresh shared random bits $b_0, \ldots, b_{127}$ and form $b = \sum_l 2^l b_l \in [0, 2^{128})$ with LIN. (Since $|p - 2^{128}|/p < 2^{-40}$, the implementation masks with 128 bits rather than with a uniform element of $[0, p)$; this is the source of the $385/p$ below.)

2. Open $\mu = (a - b) \bmod p$, as its canonical representative in $[0, p)$.

3. Compute, on bits, the wrap bit $q = [b \geq p - \mu] = [b + \mu \geq p]$, with a comparison of the shared bits of b against the public bits of $p - \mu$.

4. Let g be the public 129-bit string of μ if $q = 0$ and of $2^{129} + \mu - p$ if $q = 1$, selected with q by LIN and MULT. Add g to the bits of b with a 129-bit adder and drop the carry out of bit 128.

Correctness. Since $a \equiv b + \mu \pmod p$ and $0 \leq b + \mu < 2^{128} + p < 2p$, we have $a = b + \mu - p \cdot [b + \mu \geq p]$ as integers. The adder computes $b + \mu$ if $q = 0$, and $b + \mu - p + 2^{129} \equiv b + \mu - p \pmod{2^{129}}$ if $q = 1$; in both cases the result lies in $[0, p) \subset [0, 2^{129})$, so the 129 output bits are exactly the canonical bits of a, and bit 128 may be 1. The case $\mu = 0$ is handled separately by the implementation and gives $a = b$. A 128-bit truncation of $b + \mu$ would not compute this; the width of 129 bits is what the overflow test of §4.3 reads. We also checked the compiled routine directly: under semi-honest Shamir, malicious Shamir and MASCOT, it decomposed the boundary values $0, 1, 2, 384, 385, 2^{64}, 2^{127} - 1, 2^{127}, 2^{128} - 1, 2^{128}, 2^{128} + 1, p - 2, p - 1$, each 20 times with fresh masks, and 200 random values; all 460 outputs per protocol were canonical and recomposed to the input. Small values take the wrap branch $q = 1$ with high probability and values $\geq 2^{128}$ never do, so both branches are exercised. This is a test of the implementation, not part of the proof.

Privacy. Everything except the opening of μ is LIN and MULT on shared bits, which the perfect simulators above cover. The random bits are part of the preprocessing: in MP-SPDZ's Shamir protocols each is the XOR of bits shared by $T + 1$ different parties, at least one of them honest, or, in other configurations, is computed from the opened square of a random shared element; both are standard [11], and we include them in the realization we take as given. Each decomposition uses its own bits, so the masks of different decompositions are independent. The simulator for one decomposition is straight-line: it outputs a fresh uniform element of $\mathbb{F}_p$ as μ, and runs the perfect simulators for the remaining steps, which reveal nothing to at most T parties. For every fixed a, μ is uniform on a set of 2^{128} of the p field elements, so its distance from uniform on $\mathbb{F}_p$ is $1 - 2^{128}/p \leq 385/p$. Since the masks are independent and the rest of the corrupted view is simulated perfectly given μ, these distances add up over the q_{bd} decompositions.

We take the realization of the perfect parts of $\mathcal{F}_{\mathsf{ABB}}$ as given and do not reprove it. The claim then follows from Theorem 6.7 and the UC composition theorem [8], with the error terms adding up. MP-SPDZ's `shamir-party.x` [22], which ran the semi-honest benchmarks of §7, implements protocols of this type. □

Proof of Corollary 6.9. Fix a notion $\mathbf{X}$ of §5.6 and an adversary $\mathcal{A}$ against the real wallet. $\mathcal{A}$ corrupts a set C of at most T parties, sees their views, and chooses commands as the oracles of $\mathbf{X}$ allow: DERIVE for EXPAND, OPEN only on nodes that $\mathbf{X}$ lets it reveal, and PUB only on nodes whose public key $\mathbf{X}$ gives it. Let $\mathcal{Z}$ be the environment that runs $\mathcal{A}$ and the challenger of $\mathbf{X}$ and issues $\mathcal{A}$'s commands to all parties, in the same order, so $\mathcal{Z} \in \mathcal{Z}_{\mathrm{sync}}$. Whenever the

challenger needs a secret value, $\mathcal{Z}$ issues EXPORT of that node to a fixed honest party $P_j \notin C$, which exists since $T < N$. The *contents* of such an export are hidden from the adversary, but its *occurrence* is not: $\mathcal{F}_{\mathsf{HD}}$ forwards the command name, the node and the recipient (Definition 6.3), and the protocol's sequence of commands is public as well. $\mathcal{Z}$ therefore issues the same exports, to the same party, in the same order, for both values of the challenge bit:

- In **HD-IND** and **SI**, if the target is materialized, $\mathcal{Z}$ exports it in both branches. It uses the exported pair if $b = 0$, and discards it and uses a pair it samples itself if $b = 1$. If the target is not materialized, $\mathcal{Z}$ exports nothing, whatever b, and answers as the game does.
- In **HCU**, $\mathcal{Z}$ applies the eligibility rule of Definition 5.17, which does not depend on b. If the three candidate children are materialized, it exports u_0, u_0' and u_1 in this order and forms the challenge points itself from the exported keys. It never issues PUB on them, which would give all three points to the adversary.
- In **PCH** there is no challenge bit. As soon as w is materialized (after INIT if $w = \epsilon$, or after the DERIVE command that creates it), $\mathcal{Z}$ issues PUB(w) and records P_w. After $\mathcal{A}$ outputs its guess k^*, $\mathcal{Z}$ checks locally whether w was materialized and $k^* \cdot G = P_w$, with no further protocol command. (If $\mathcal{A}$ later queries PUB(w), $\mathcal{Z}$ forwards it as usual.)

Hence the public sequence of commands, and with it the leakage $\mathcal{S}$ receives from $\mathcal{F}_{\mathsf{HD}}$, is a function of $\mathcal{A}$'s queries and the verdicts only, and does not depend on b. $\mathcal{Z}$ outputs 1 if $\mathcal{A}$ wins.

By Corollary 6.8, the probability that $\mathcal{Z}$ outputs 1 changes by at most δ_{UC} when the real execution is replaced by the ideal one with $\mathcal{F}_{\mathsf{HD}}$ and $\mathcal{S}$. In the ideal execution, $\mathcal{A}$'s view is the output of $\mathcal{S}$, which is computed only from the public commands, from λ, and from the values $\mathcal{F}_{\mathsf{HD}}$ releases to the adversary by OPEN and PUB (an EXPORT to the honest P_j releases its name, node and recipient, and nothing else). The commands are those $\mathcal{A}$ chose, together with the exports above and, in **PCH**, the command PUB(w), which $\mathcal{A}$ can predict from its own queries and the verdicts. The other values are exactly the answers of the oracles EXPAND and REVEAL and the public keys that $\mathbf{X}$ provides. Hence $\mathcal{A}$ run together with $\mathcal{S}$, with the exports inserted into $\mathcal{S}$'s input by the game adversary itself, is an adversary against $\mathbf{X}$ as defined in §5; it wins with exactly the ideal-execution probability, and Theorem 5.29 bounds that probability. For **PCH** the advantage is the winning probability, so the loss is δ_{UC}. For the decisional notions the advantage is $|2\Pr[\mathrm{win}] - 1|$, so a change of δ_{UC} in the winning probability changes the advantage by at most $2\delta_{\mathrm{UC}}$. □

D.4 Malicious adversaries

This subsection proves the malicious results of §6.7, Theorem 6.12 and Corollary 6.14.

Proof of Theorem 6.12. What a malicious party can do. In the $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$-hybrid model every value of the derivation lives inside the black box, and the parties have no private inputs: the root comes from ROOTGEN, and every command from $\mathcal{Z}$ carries only public data (§6.1). A party's only actions are the commands it sends to $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$. The honest parties send exactly the commands of Π_{HD}, so a corrupted party that sends anything else causes an abort (Definition 6.10), and a corrupted party that withholds a command makes $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$ wait until it sends it. A malicious adversary can therefore do exactly three things beyond what a semi-honest one can: stop the execution by deviating, stop it by sending abort, or delay it by withholding a command. In the first two cases every party receives $\perp$; in the third no party receives an output and no value is released while the command is withheld. It cannot make the black box compute a wrong value or release a value the protocol does not open.

The simulator. $\mathcal{S}^{\mathrm{abort}}$ runs $\mathcal{A}$ and handles one command at a time. It receives from $\mathcal{F}_{\mathsf{HD}}^{\mathrm{abort}}$ the command and the adversary's values, and replays the $\mathcal{F}_{\mathsf{ABB}}^{\mathrm{abort}}$ commands of Π_{HD} for it step by step, producing every released value exactly as the simulator of §D.1 does: fresh uniform

values for the cube openings, the bits of λ for the overflow verdicts, 0 or a fresh uniform nonzero value for the masked validity values, and the values received from $\mathcal{F}_{\mathsf{HD}}^{\text{abort}}$ for OPEN, PUB and EXPORT. At each step it checks the commands $\mathcal{A}$ sends on behalf of the corrupted parties. If $\mathcal{A}$ deviates from Π_{HD} or sends abort, during a command or between two commands, $\mathcal{S}^{\text{abort}}$ sends abort to $\mathcal{F}_{\mathsf{HD}}^{\text{abort}}$ and stops. While $\mathcal{A}$ withholds the command of the current step for a corrupted party, $\mathcal{S}^{\text{abort}}$ waits as well and sends nothing to $\mathcal{F}_{\mathsf{HD}}^{\text{abort}}$, which then gives the honest parties no output either; when the command arrives, the simulation goes on with that step. After the last step of the command it sends continue.

Indistinguishability. Fix $\mathcal{A}$ and $\mathcal{Z}$. Up to the first deviation or abort, the real execution is an execution of Π_{HD} in which every party follows the protocol, possibly with delays, because $\mathcal{F}_{\mathsf{ABB}}^{\text{abort}}$ then behaves exactly as $\mathcal{F}_{\mathsf{ABB}}$. By the hybrids of Theorem 6.7, the sequence of values released to $\mathcal{A}$ up to any point is distributed, jointly with the honest parties' outputs so far, as the sequence $\mathcal{S}^{\text{abort}}$ produces, up to the event that some mask ρ is zero, of probability at most q_{att}/p. Whether and when $\mathcal{A}$ deviates, aborts or withholds a command is a function of its view and its randomness only, so it has the same distribution in both executions. While a command is withheld, neither execution releases a value or gives an honest party an output. After that point both executions end with $\perp$ for every party and release nothing more. Before it, every completed command gives the honest parties the same outputs in both executions, by Lemma 6.6. Hence $\mathcal{Z}$ distinguishes the two executions with advantage at most q_{att}/p. □

Proof of Corollary 6.14. As for Corollary 6.9, with Corollary 6.13 in place of Corollary 6.8 and $\mathcal{S}^{\text{abort}}$ in place of $\mathcal{S}$; $\mathcal{Z}$ issues the same b-independent exports during TEST for the decisional notions, records P_w via PUB(w) upon materializing w for **PCH**, and verifies $\mathcal{A}$'s final output locally without issuing any command after $\mathcal{A}$ halts. The factor 2 for the decisional notions arises in the same way. If the execution aborts, or if $\mathcal{A}$ halts or returns control without letting an outstanding command complete because a corrupted party withholds its message, $\mathcal{Z}$ outputs a fresh uniform bit for the decisional notions and 0 (failure) for **PCH**. By Corollary 6.13, the probability that $\mathcal{Z}$ outputs 1 changes by at most $q_{\text{att}}/p + \varepsilon_{\text{ABB}}$ when passing from the real execution to the ideal one with $\mathcal{F}_{\mathsf{HD}}^{\text{abort}}$ and $\mathcal{S}^{\text{abort}}$. In the ideal execution, $\mathcal{A}$'s view is produced by $\mathcal{S}^{\text{abort}}$ from the values that $\mathcal{F}_{\mathsf{HD}}^{\text{abort}}$ releases to the adversary, which are the answers of the oracles of **X** (including PUB(w) in **PCH**, which Definition 5.12 permits). An abort or a stalled command corresponds to the standalone adversary $\mathcal{A}$ running with $\mathcal{S}^{\text{abort}}$ stopping its queries and outputting a fresh uniform bit for the decisional notions or $\perp$ for **PCH**. Hence this composed adversary is valid against **X**, wins with exactly the ideal-execution probability, and Theorem 5.29 bounds its advantage. □

Remark D.1 (MASCOT, and the related-key argument). MASCOT [23], which we also ran (§7.6), is designed to realize the linear, multiplication and opening commands of $\mathcal{F}_{\mathsf{ABB}}^{\text{abort}}$ against static malicious adversaries corrupting up to $N-1$ parties: any deviation is caught by the information-theoretic MACs of SPDZ-style sharing and makes the honest parties abort. Theorem 6.12 would apply to it given a realization of the remaining commands. Our results for MASCOT are limited to two parties, and we have not examined MP-SPDZ's MASCOT code. Malicious security also bears on the designers' related-key argument. HYDRA's designers exclude related-key attacks on two grounds [20, §5.5]: keys are sampled at random for each computation, so an attacker can neither know nor choose relations between them; and in an actively secure protocol any difference added to a shared key is detected. The first ground does not hold for derived child keys, which are computed from their parent rather than freshly sampled. The second applies to Π_{HD} under malicious Shamir sharing and MASCOT, not under semi-honest Shamir sharing, where a party that deviates is outside the model.

E Additional measurements

This appendix collects the tables that §7 refers to.

Table 4: Time per part of one derivation in seconds, $N = 5$, $T = 2$, under malicious Shamir sharing and, for Hydra, under semi-honest Shamir sharing (medians of 20 runs without delay and 5 at 100 ms). Percentages are shares of the median total time, computed from unrounded medians, so they can differ by one point from a ratio of the rounded entries. Each part includes the preprocessing batches that are first requested in it, so the split without delay depends on the order of the program. Each part of HMAC-SHA512 but the last is two SHA-512 compressions; HMAC-SHA512 was not measured at 100 ms (Table 3).

		Malicious Shamir		Semi-honest Shamir	
	Part	RTT 0	RTT 100 ms	RTT 0	RTT 100 ms
Hydra	PRF E_K (§2.2)	0.080 (24%)	7.18 (51%)	0.081 (53%)	13.7 (71%)
	overflow test (§4.3)	0.020 (6%)	1.64 (12%)	0.015 (10%)	1.76 (9%)
	bits of k_{par} (§4.4)	0.149 (45%)	3.57 (25%)	0.047 (31%)	2.70 (14%)
	add mod n, verdict (§4.4)	0.076 (23%)	1.57 (11%)	0.009 (6%)	1.12 (6%)
HMAC-SHA512	padded keys, per parent	1.59 (48%)			
	inner, outer hash, per child	1.66 (50%)			
	add mod n, verdict	0.09 (3%)			

Table 5: Time of one derivation under a bandwidth limit shared by all parties, $N = 5$, $T = 2$, no added delay. The first two columns use malicious Shamir sharing (medians of 3 and 5 runs); the last gives Hydra under semi-honest Shamir sharing for reference (median of 5 runs).

	Malicious Shamir			Semi-honest
Limit	HMAC-SHA512	Hydra	Ratio	Hydra
1000 Mbit/s	5.73 s	1.04 s	5.5×	0.35 s
100 Mbit/s	30.7 s	9.12 s	3.4×	2.79 s
10 Mbit/s	311.9 s	92.2 s	3.4×	28.3 s

Table 6: Effect of the number of parties N and the threshold T, no network delay. The HMAC-SHA512 and first HYDRA columns use malicious Shamir sharing (medians of 3 and 10 runs); the last column gives HYDRA under semi-honest Shamir sharing for reference (median of 10 runs). Rounds and data are those of party P_1. These runs are separate from those of Table 2, so the $N = 5$, $T = 2$ row differs from it by run-to-run variation.

		Malicious Shamir					Semi-honest
		HMAC-SHA512		HYDRA			HYDRA
N	T	Time	Data	Time	Rounds	Data	Time
3	1	1.80 s	28.6 MB	0.116 s	242	11.3 MB	0.064 s
5	1	2.19 s	36.3 MB	0.180 s	242	15.6 MB	0.089 s
5	2	3.31 s	53.4 MB	0.329 s	243	22.1 MB	0.155 s
7	1	2.82 s	44.0 MB	0.247 s	242	19.9 MB	0.133 s
7	2	4.46 s	60.3 MB	0.438 s	243	26.6 MB	0.230 s
7	3	6.23 s	77.2 MB	0.680 s	244	33.5 MB	0.348 s
9	4	10.28 s	100.7 MB	1.274 s	245	45.8 MB	0.687 s

Table 7: One hardened derivation under MASCOT, $N = 2$ (medians of 5 runs without delay; with delay, of 5 runs for HMAC-SHA512 and 3 for HYDRA). Rounds and data are those of party P_1.

	HMAC-SHA512	HYDRA	Ratio
Time, no delay	1.95 s	3.30 s	0.59×
Time, RTT 1 ms	25.4 s	3.66 s	6.9×
Time, RTT 10 ms	174.9 s	7.37 s	24×
Communication rounds	31,374	660	48×
Data sent	510 MB	443 MB	1.2×
CPU time	1.60 s	3.29 s	
Peak memory	78 MB	197 MB	

Table 8: Communication of one HYDRA derivation by type of message, for party P_1, $N = 5$, $T = 2$, from MP-SPDZ's verbose statistics (three runs each, with identical counts).

	Semi-honest Shamir		Malicious Shamir	
Type of message	Rounds	Data	Rounds	Data
Openings	135	0.01 MB	216	2.4 MB
Resharing and preprocessing	216	6.7 MB	17	12.5 MB
Broadcasts (checks)			10	< 0.01 MB
Not attributed by the breakdown		0.02 MB		7.2 MB
Total	351	6.75 MB	243	22.1 MB
Triples used		148		10,120