sciendo

Proceedings on Privacy Enhancing Technologies ; 2021 (4):76–95

Konstantinos Chalkias*, Shir Cohen, Kevin Lewi, Fredric Moezinia, and Yolan Romailler

HashWires: Hyperefficient Credential-Based Range Proofs

Abstract: This paper presents HashWires, a hash-based range proof protocol that is applicable in settings for which there is a trusted third party (typically a credential issuer) that can generate commitments. We refer to these as "credential-based" range proofs (CBRPs). HashWires improves upon hashchain solutions that are typically restricted to micro-payments for small interval ranges, achieving an exponential speedup in proof generation and verification time. Under reasonable assumptions and performance considerations, a HashWires proof can be as small as 305 bytes for 64-bit integers. Although CBRPs are not zero-knowledge and are inherently less flexible than general zero-knowledge range proofs, we provide a number of applications in which a credential issuer can leverage HashWires to provide range proofs for private values, without having to rely on heavyweight cryptographic tools and assumptions.

Keywords: range proofs, credentials, hash-chains, accumulators, cryptographic commitments, malleability, micro-payments, location privacy

DOI 10.2478/popets-2021-0061
Received 2021-02-28; revised 2021-06-15; accepted 2021-06-16.

1 Introduction

A range proof is a special type of zero-knowledge proof (ZKP) [26] for proving that a given integer lies within an interval, without revealing anything else about the integer. Maxwell used them to achieve confidential transactions [31], where an amount is hidden in a Pedersen commitment, and a range proof allows a verifier to check that it lies within a non-negative interval. Range proofs were first conceived in works by Damgård [19] and Fujisaki and Okamoto [24], with more practical constructions from Boudot [5], Camenisch et al. [11], and most notably, Bulletproofs by Bünz et al. [10]. In this work, we focus specifically on such range proofs over committed values.

A typical zero-knowledge range proof (ZKRP) over a committed value involves a cryptographic commitment and a proof. The commitment must satisfy conditions known as hiding and binding, and the proof must satisfy completeness, soundness, and zero-knowledge. ZKRPs have been used to construct private transactions on blockchains (such as Maxwell's Confidential Transactions proposal [31] and MimbleWimble [36]) and privacy-preserving smart contracts on Ethereum. They have also been proposed for handling proofs of solvency [15, 18], privacy for electronic voting and auction systems, and anonymous credentials (see [33] for a more extensive survey of ZKRP applications).

Some applications of ZKRPs exist in the setting in which a trusted entity acts as an authority for either producing or validating commitments over integer values.[1] As an illustrative example, consider the setting in which a citizen wishes to prove to a service provider that their age is above a certain threshold, without having to reveal their birth date [8, 20]. The trusted entity (e.g., a governmental identification provider) can issue a statement committing to the prover's age, and the prover has the capability of using a ZKRP protocol to produce a proof that convinces a verifier that their age is indeed above the requested threshold.

Although the above scenario could be solved with existing ZKRP constructions, such as Bulletproofs or Pedersen commitments, these solutions are often less practical in resource-constrained environments where asymmetric-key cryptography operations are considered expensive. In applications where these proofs are to be generated and verified very frequently, focusing on implementing simpler, light-weight primitives can be an

***Corresponding Author: Konstantinos Chalkias, Kevin Lewi, Fredric Moezinia:** Novi / Facebook, E-mail: {kostascrypto,klewi,moezinia}@fb.com
Shir Cohen: Novi / Facebook / Technion, E-mail: shirco@cs.technion.ac.il
Yolan Romailler: Novi / Facebook / Kudelski Security, E-mail: yolan@romailler.ch

1 Not to be confused with "trusted setup", which requires trust in the authority to correctly set up parameters that are critical to the security of the range proof itself.

attractive alternative. In this work, we are motivated by the following question:

"Can we construct more efficient range proofs for settings in which there is already a trusted party that generates the commitment?"

In particular, can we leverage the assumption that the verifier of the range proof can trust that the commitment was generated honestly, in order to yield simpler and more efficient constructions?

To capture the additional requirement of a trusted "issuer" of the commitment, we introduce *credential-based range proofs* (CBRPs), which differ from ZKRPs in two ways: the soundness requirement is relaxed to a weaker notion which we refer to as *commitment-conditional soundness*, and the zero-knowledge property is relaxed to *witness indistinguishability*. We provide a formal treatment of these security definitions in Section 2.1.

Hashchains. To help intuitively understand these relaxed security requirements for CBRPs, we begin with the following description of a simple hash-based scheme for range proofs, originally presented by Rivest and Shamir [37] in a protocol known as PayWord (with adaptations). Given a collision-resistant hash function H and an integer N representing the maximum possible value of the domain, the issuer produces a commitment c for a secret integer $k \in [0, N]$ by selecting a random seed r and setting $c = \mathsf{H}^k(r)$ (using $\mathsf{H}^k(\cdot)$ to represent k repeated iterations of the function H) and publishing c to a public bulletin board (or sign and send c to the prover directly). The issuer sends the randomness r and integer k to the prover, who can then produce a range proof over r for the threshold t by computing $\pi = \mathsf{H}^{k-t}(r)$ and sending π to the verifier. The verifier checks that $\mathsf{H}^t(\pi) = c$ to be convinced that c is a commitment of some value $x \geq t$.

Despite its simplicity, PayWord suffers from a significant drawback: letting $n = \lceil \log_2 N \rceil$ be the number of bits to represent the size of the domain N, note that commitment and proof generation, and verification are all asymptotically exponential in n. Thus, using PayWord as a CBRP is really only suitable for small domains, and its performance for large ranges is unlikely to be competitive against generic ZKRPs.

Using garbled circuits. A completely different approach for building CBRPs for large domains without using ZKRPs involves the use of garbled circuits. For a secret integer $k \in [0, N]$, the issuer can produce a commitment representing a hash of a garbling of the circuit C that evaluates as $C(k, y) = 1$ if $k \geq y$ and 0 otherwise, along with commitments to the randomized values associated with each of the circuit's input wires. The prover, with an input threshold t, produces a proof which consists of the garbled circuit for C, along with an opening of the randomized values corresponding to the input wires for the bit representation of t. To verify this proof, a verifier can check that the opened commitments correspond to the correct bits of t, that the hash of the garbled circuits corresponds to what is in the final commitment, and that the evaluation of the garbled circuit produces the expected outcome.

Intuitively, the commitment-conditional soundness derived from the correctness of the garbling scheme, and witness indistinguishability follows from the privacy of the garbled circuit itself. This solution is clearly generalizable to arbitrary circuits (instead of just being restricted to handle range proofs), but the size of a proof in this scheme is quite large, mainly due to the fact that it must contain the entire description of a garbled circuit. Therefore, while this approach can technically handle the large-domain inputs that PayWord could not, there is still a substantial amount of room for improvement.

1.1 Our Contributions

In this work, we propose HashWires, which completely overcomes the small-domain limitations of PayWord. In particular, for n-bit intervals, HashWires achieves commitment generation, proving/verification time, and proof size to simultaneously be *linear* in n, while still relying only on the existence of collision-resistant and random-output hash functions (defined in Section 2.2).

The central component of the HashWires construction, described in Section 3, relies on a simple combinatorial observation about a decomposition of any positive integer x into a set of integers with a certain structure that is naturally amenable to each of its elements being represented by a sequence of hash chains. We call this set a minimum dominating partition (MDP), defined in Definition 9. We then convert the elements of this set into separate groups of hash *multichains* in order to achieve an exponential speedup in both proving and verification time, taking extra care to ensure that the witness indistinguishability property is still preserved.

We also provide a concrete instantiation of HashWires, described in Section 4. This instantiation includes a number of performance optimizations and prac-

tical considerations on top of the main construction of Section 3, and is incorporated into an open-source Rust implementation of HashWires. Another interesting property is that HashWires can be implemented with any hash function and hashchain sizes, enabling compatibility and proof-size versus computational efficiency trade-offs. We also highlight that most blockchains do not offer support for elliptic curve operation primitives in their smart contracts, but many allow hash function invocations (i.e., KECCAK-256 in Ethereum [29, 43] and {SHA2, SHA3}-256 in Diem [35]) which enables implementing HashWires verification in smart contracts without any modification to the underlying blockchain virtual machine.

Furthermore, we offer benchmarks for various implementation instances, along with a comparison in proof size and cost against existing works (most notably, Bulletproofs) that achieve general range proofs, taking advantage of the described optimizations and our weaker security model where the commitment is assumed to be constructed honestly. In particular, for 64-bit domains, a reasonably optimized HashWires proof (using 32-byte hash outputs and hashchains consisting of 256 elements) can be just *369 bytes long*. Interestingly, using 2048-long chains results to 177-byte proofs for ranges up to 2^{32}. As a matter of reference, Bulletproofs produce range proofs of 692 bytes and are orders of magnitude slower in terms of prover and verifier time, while Groth16 [28] outputs fixed-size proofs of 192 bytes. It is highlighted though that a direct comparison against ZKPR schemes would be unfair as these systems clearly operate in a stronger security model.

In Section 5, we also show how the structure of HashWires commitments can be used to support conjunctions of range queries, effectively extending the scope of applications that these range proofs can be applied to. Additionally, we identify proof malleability issues on the original PayWord scheme and apply an efficient fix inspired by the Winternitz checksum trick used in hash-based signatures [9, 22, 32]. We also provide a number of applications which can be addressed using CBRPs.

1.2 Related Work

In [37], the PayWord scheme uses hash chains in order to provide a micro-payment solution, where the hashchain proves possession of a given monetary value, allowing an individual to spend up to a given amount only with knowledge of a seed. The PayWord scheme was expanded to work with multiple monetary denominations [44] using multiple hashchains, allowing to represent larger values more easily, but without the asymptotic improvement that HashWires achieves. Later, in [30], a generalization of PayWord was proposed which improves verification time, but still uses exponential time for commitment generation. PayWord's approach has also been used in auctions and proving age.

There is of course a large amount of prior work in the study of zero knowledge proofs, as well as range proofs specifically [5, 10, 11, 19, 24]. The goal of our work is to further improve efficiency and proof size for these schemes for specific scenarios by leveraging the assumption of a trusted entity that can securely generate the commitment.

Anonymous credentials. Another line of work which bears resemblance to CBRPs includes the study of anonymous credentials (AC) [1, 3, 6], described in [17] and realized in [12]. In the anonymous credentials setting, there is also the notion of a trusted issuer which produces a credential for a user. The user can then present this credential to different verifiers, who can check its validity. Two notable security properties which have been focused on in this setting are the "unlinkability" and "anonymity" of credentials. The first property states that a collusion between any combination between verifiers and the issuer cannot link a) two or more shows of the same credential and b) an issued and a presented credential. The anonymity property usually refers to blindness when signing credentials, but also the selective disclosure property for the attributes associated with the credential (which allows a verifier to check that the credential satisfies some notion of validity without learning any extra information about the credential).

HashWires can be viewed as having applications to anonymous credentials by providing the anonymity / selective-disclosure property (albeit weakened to witness indistinguishability instead of zero knowledge), without the focus on satisfying the unlinkability property for the credential issuer. We also highlight that some AC schemes are single use, i.e., the ACL scheme in [3], while others can be efficiently used many times [12]. An interesting future direction for HashWires would be to support the unlinkability property as well, potentially by accompanying it with a zero knowledge proof of correct commitment construction and proof of issuance.

2 Preliminaries

In this work, we refer to λ as the security parameter. For an integer N, we use $[N]$ to represent the set of integers $\{1, 2, \ldots, N\}$. For two integers $a < b$, we use $[a, b]$ to represent the set of integers $\{a, a+1, ..., b\}$. We use the terms "efficient" to represent an algorithm that runs in time polynomial in λ, and "negligible" to denote a quantity that is inversely proportional to being polynomial in λ. For two distributions D_1 and D_2 over the same domain, we write $D_1 \approx D_2$ to represent computational indistinguishability; namely, for every efficient adversary that receives as input a sample and outputs a bit, the difference in probability that the adversary outputs 1 when receiving a sample from D_1 versus the probability of the adversary outputting 1 when receiving a sample from D_2 is negligible in λ. We refer to the above quantity as the *advantage* of an adversary in distinguishing D_1 from D_2.

We use $\{0,1\}^*$ to represent bit strings of arbitrary length. We use $||$ to represent bit string concatenation. For an integer n, we use $\langle a_1, \ldots, a_n \rangle$ to denote an ordered sequence of a set of elements $\{a_1, \ldots, a_n\}$.

We use the notation $y_{i,1} \cdots y_{i,d}$ to represent the base-b representation with d digits of the integer y_i.

2.1 Credential-Based Range Proofs

For a fixed positive integer N, along with a commitment space $\mathbb{C}$, proof space $\mathbb{P}$, and randomness space $\{0,1\}^\lambda$, a non-interactive CBRP is a tuple of algorithms $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Prove}, \mathsf{Verify})$ with the following properties:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$. On input the security parameter λ, the setup algorithm outputs public parameters pp.
- $\mathsf{Commit}(\mathsf{pp}, x; r) \to \mathsf{com}$. On input the public parameters pp, an integer $x \in [N]$ and randomness $r \in \{0,1\}^\lambda$, the commit algorithm outputs a commitment $\mathsf{com} \in \mathbb{C}$.
- $\mathsf{Prove}(\mathsf{pp}, x, t; r) \to \pi$. On input the public parameters pp, an integer $x \in [N]$, a threshold $t \in [N]$, and randomness $r \in \{0,1\}^\lambda$, the prove algorithm outputs a proof π.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{com}, t, \pi) \to z$. On input the public parameters pp, a commitment $\mathsf{com} \in \mathbb{C}$, a threshold $t \in [N]$, and a proof π, the verify algorithm outputs a bit $z \in \{0,1\}$.

Security. The following security properties of a credential-based range proof over a commitment are inherited from the typical security properties for any commitment scheme (hiding and binding) as well as any range proof protocol (completeness, soundness, and zero knowledge). These properties are similar to (and mirrored from) a zero-knowledge range proof (ZKRP), except for the definition of soundness, and a relaxation from zero knowledge to witness indistinguishability.

Definition 1 (Binding Commitment)**.** The commitment for a CBRP is binding if for all efficient adversaries $\mathcal{A}$ that receive as input a set of public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and output a pair of distinct integers $x_0, x_1 \in [N]$ along with a pair of random strings $r_0, r_1 \in \{0,1\}^\lambda$, with $\mathsf{com}_0 \leftarrow \mathsf{Commit}(\mathsf{pp}, x_0; r_0)$ and $\mathsf{com}_1 \leftarrow \mathsf{Commit}(\mathsf{pp}, x_1; r_1)$, the quantity $\Pr[\mathsf{com}_0 = \mathsf{com}_1]$ is negligible in λ.

Definition 2 (Perfect Completeness)**.** A CBRP satisfies (perfect) completeness if for every adversary $\mathcal{A}$ that outputs an $x \in [N]$, $t \in [N]$ where $x \geq t$, and $r \in \{0,1\}^\lambda$, it always holds that $\mathsf{Verify}(\mathsf{Commit}(x; r), t, \mathsf{Prove}(x, t; r)) = 1$.

Definition 3 (Commitment-Conditional Soundness)**.** A CBRP satisfies commitment-conditional soundness if for every adversary $\mathcal{A}$ that takes as input the public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and outputs a tuple $(x, t, r, \pi) \in [N] \times [N] \times \{0,1\}^\lambda \times \mathbb{P}$ for which $x < t$, the probability that $\mathsf{Verify}(\mathsf{pp}, \mathsf{Commit}(\mathsf{pp}, x; r), t, \pi) = 1$ is negligible in λ.

Note that this definition is weaker than the usual definition of soundness for a range proof system. Typically, for soundness, the adversary is allowed to produce a commitment com, along with the threshold t and proof π, and the adversary wins if $\mathsf{Verify}(\mathsf{com}, t, \pi) = 1$ and there does not exist an $(x, r) \in [N] \times \{0,1\}^\lambda$ such that $\mathsf{com} = \mathsf{Commit}(x; r)$ and $x \geq t$. In our case, however, we are restricting the adversary's power by ensuring that the verifier cannot receive invalid commitments.

Definition 4 ($\mathsf{Expt}^{\mathsf{WI}}$)**.** For an integer N, the experiment $\mathsf{Expt}_b^{\mathsf{WI}}(\mathcal{A})$ between a challenger and an adversary $\mathcal{A}$ proceeds as follows:

1. The challenger sends $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends to the challenger $x, t \in [N]$, where $x \geq t$.
3. The challenger samples randomness $r \leftarrow_{\mathsf{R}} \{0,1\}^\lambda$ and returns $(\mathsf{com}_b, \pi_b) \leftarrow (\mathsf{Commit}(x_b; r), \mathsf{Prove}(x_b, t; r))$ to $\mathcal{A}$.

4. $\mathcal{A}$ outputs a bit, which is also the output of the experiment.

Definition 5 (Witness Indistinguishability)**.** A CBRP satisfies witness indistinguishability if for all efficient adversaries $\mathcal{A}$, the quantity $\Pr[\mathsf{Expt}_0^{\mathsf{WI}}(\mathcal{A}) = 1] - \Pr[\mathsf{Expt}_1^{\mathsf{WI}}(\mathcal{A}) = 1]$ is negligible in λ.

Finally, we say that a CBRP is secure if all of the above properties: binding commitment, completeness, soundness, and witness indistinguishability, hold. Traditional commitment schemes also require a "hiding" property, but we omit this definition since it is actually implied by the witness indistinguishability property of the proof, which is a strictly stronger notion.

2.2 Random-Output Hash Functions

In this work, we consider hash functions which satisfy random-output security.

Definition 6 (Random-Output Hash Function)**.** Let U be the uniform distribution over $\{0,1\}^\lambda$. For an integer d, let $H_d(U)$ represent the distribution defined as $\{H(x) : x \leftarrow_{\mathsf{R}} \{0,1\}^d\}$. We say that a hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$ is *random-output secure* if $H_d(U) \approx U$ for all $d \geq \lambda$.

Any hash function which can be modeled after a random oracle is also random-output secure. However, note that random-output security and collision resistance for hash functions are incomparable notions of security. For a collision-resistant function $G : \{0,1\}^* \to \{0,1\}^{\lambda-1}$, the function H defined as $H(x) \coloneqq 0 \,||\, G(x)$ is also collision-resistant, but not random-output secure. Similarly, for a random-output secure function $F : \{0,1\}^* \to \{0,1\}^\lambda$, the function H defined as

$$H(x) \coloneqq \begin{cases} 0, & \text{if } x = 0^\lambda \text{ or } x = 0^{\lambda+1} \\ F(x), & \text{otherwise} \end{cases}$$

is also random-output secure, but not collision-resistant.

Pseudorandom generators. Recall that a pseudorandom generator [25] is a function $G : \mathcal{X} \to \mathcal{Y}$ (where $|\mathcal{Y}| > |\mathcal{X}|$) with

$$\{G(x) : x \leftarrow_{\mathsf{R}} \mathcal{X}\} \approx \{y : y \leftarrow_{\mathsf{R}} \mathcal{Y}\}.$$

In other words, the distribution induced by $G(x)$ for randomly chosen $x \leftarrow_{\mathsf{R}} \mathcal{X}$ is indistinguishable from y for randomly chosen $y \leftarrow_{\mathsf{R}} \mathcal{Y}$.

2.3 Accumulators

For a domain $\mathcal{X}$ and output space $\mathcal{Y}$, an accumulator $\mathsf{ACC} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{WitCreate}, \mathsf{Verify})$ is a tuple of four algorithms defined as follows:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$. The setup algorithm takes as input the security parameter λ and outputs public parameters pp.
- $\mathsf{Eval}(\mathsf{pp}, S) \to \mathsf{acc}$. The evaluation algorithm takes as input the public parameters pp and a set $S \subseteq \mathcal{X}$, and produces an accumulation acc.
- $\mathsf{WitCreate}(\mathsf{pp}, x, S) \to \mathsf{wit}$. The witness creation algorithm takes as input the public parameters pp, an element $x \in \mathcal{X}$, and a subset $S \subseteq \mathcal{X}$, and produces a witness wit.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{acc}, \mathsf{wit}, x) \to b$. The verification algorithm takes as input the public parameters pp, an accumulation $\mathsf{acc} \in \mathcal{Y}$, a witness wit, and an input $x \in \mathcal{X}$, and outputs a bit $b \in \{0,1\}$ representing whether or not verification succeeded.

Using the terminology from [21], we consider static accumulators that are non-universal (supporting only inclusion proofs), and deterministic. We also alter the witness creation algorithm by allowing it to take as input a set S instead of the accumulation along with an auxiliary string (as done in [21]). Although this is a slight loss of generality, it simplifies the presentation of our main construction without affecting the functionality or security of the protocol.

Correctness. An accumulator is correct if, for $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, for any set $S \subseteq \mathcal{X}$ with $x \in S$, it holds that

$$\mathsf{Verify}(\mathsf{pp}, \mathsf{Eval}(\mathsf{pp}, S), \mathsf{WitCreate}(\mathsf{pp}, x, S), x) = 1.$$

Security. We define security for accumulators as follows:

Definition 7 ($\mathsf{Expt}^{\mathsf{ACC}}$)**.** For an adversary $\mathcal{A}$, we define $\mathsf{Expt}^{\mathsf{ACC}}(\mathcal{A})$ with a challenger as follows:

1. The challenger sends $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends a witness wit, an element $x \in \mathcal{X}$, and a set $S \subseteq \mathcal{X}$ to the challenger.
3. The challenger computes $\mathsf{acc} \leftarrow \mathsf{Eval}(\mathsf{pp}, S)$, and the output of the experiment is 1 if $x \notin S$ and $\mathsf{Verify}(\mathsf{pp}, \mathsf{acc}, \mathsf{wit}, x) = 1$, and 0 otherwise.

Definition 8 (Secure Accumulator [21])**.** An accumulator ACC is secure if for all efficient adversaries $\mathcal{A}$, the quantity $\Pr[\mathsf{Expt}^{\mathsf{ACC}}(\mathcal{A}) = 1]$ is negligible in λ.

2.4 Minimum Dominating Partitions

In this section, we provide the building blocks of the HashWires construction in Section 3, beginning with the definition of a minimum dominating partition.

Definition 9 (Minimum Dominating Partition)**.** For a base b and two positive integers $x, y \in \mathbb{Z}^+$, let $x = x_1 x_2 \cdots x_d$ and $y = y_1 y_2 \cdots y_d$ represent their corresponding base decomposition into digits, with $d = \lceil \log_b(\max(x, y) + 1) \rceil$. We say that x "dominates" y if for every $i \in [d]$, $x_i \geq y_i$ [4].

For a base b and positive integer $x \in \mathbb{Z}^+$, we define the "dominating partition", denoted $\mathsf{DP}_b(x)$, to be an ordered sequence of integers that satisfy the following properties:

- The maximum element of $\mathsf{DP}_b(x)$ is x.
- For every $y \in [0, x]$, there is an element $z \in \mathsf{DP}_b(x)$ for which z dominates y in base b.

We write $\mathsf{MDP}_b(x)$ to represent a minimum dominating partition of x in base b: a dominating partition that achieves the minimum size across all valid dominating partitions of x in base b.

Examples. The sequence $\langle 3413, 3409, 3399, 2999 \rangle$ is a valid dominating partition of 3413 in base 10 (in fact, it is also minimum). In particular, all integers in $[0, 2999]$ are dominated by 2999, all integers in $[3000, 3399]$ are dominated by 3399, all integers in $[3400, 3409]$ are dominated by 3409, and all integers in $[3410, 3413]$ are dominated by 3413. Note that some integers can be dominated by more than one element of the sequence. For instance, the integer 1412 is dominated by 2999 and 3413. But it is not dominated by 3399 because of its second digit $4 > 3$, and it is not dominated by 3409 because of its third digit $1 > 0$.

As counterexamples, note that $\langle 3413, 3399, 2999 \rangle$ is not a valid dominating partition of 3413 in base 10 since the integers in the range $[3404, 3409]$ are not dominated by any integer in that sequence. Also note that while $\langle 1999, 999, 1555 \rangle$ is a dominating partition for the value 1999 in base 10, it is not minimum: $\mathsf{MDP}_{10}(1999) = \langle 1999 \rangle$. Finally, while 3999 dominates all integers in $[0, 2999]$, it also dominates the integers $[3000, 3999]$ and therefore is a dominating partition for 3999, but not for 2999.

Computing MDPs efficiently. In general, constructing a minimum dominating partition for an integer $x \in [N]$ is efficiently computable and straightforward—see Algorithm in Fig. 1 for a complete description of the procedure.

Fig. 1. Algorithm: Computing $\mathsf{MDP}_b(x)$

```
Let d ← ⌈log_b(x + 1)⌉
Initialize σ ← [x]
for i from 1, ..., d − 1 do
    if (x + 1) mod b^i ≠ 0 then
        y_i ← ⌊x/b^i⌋ · b^i − 1
        σ ← σ ∪ {y_i}
    end if
end for
return σ
```

For any two integers x and b, let $\sigma \leftarrow \mathsf{MDP}_b(x)$ be the output of the algorithm. First, we show that σ is a dominating partition for base b and integer x, and then we show that it must be of minimum cardinality. We start with some auxiliary lemmas, and show that σ does not have redundancy (its own elements do not dominate each other).

In the following, we use $d \leftarrow \lceil \log_b(x + 1) \rceil$. We also define $y_0 \leftarrow x$, and for each $i \in [d - 1]$, we set $y_i \leftarrow \lfloor x/b^i \rfloor \cdot b^i - 1$ (the value added to σ at the i^{th} iteration of Algorithm 1).

Lemma 1. *For each $i \in [d - 1]$, x dominates y_i if and only if $(x + 1) \bmod b^i = 0$.*

Proof. Note that by construction, the digits $y_{i,d-i+1} \cdots y_{i,d}$ are all equal to the digit $b - 1$, and $y_{i,d-i} \equiv x_{d-1} - 1 \pmod b$, and the remaining digits are identical between x and y_i. We note that x dominates y_i if and only if $x_{d-i+1} \cdots x_d$ are equal to the digit $b-1$ as well. Hence, x dominates y_i if and only if $y_i + b^i = x$. Setting y_i we get $\lfloor x/b^i \rfloor \cdot b^i - 1 + b^i = b^i \left(\lfloor x/b^i \rfloor + 1 \right) - 1 = x$, which completes the proof. □

Lemma 2. *For any two distinct elements $z_1, z_2 \in \sigma$, z_1 does not dominate z_2.*

Proof. Assume that there exists a pair $z_1, z_2 \in \sigma$ such that $z_1 \neq z_2$ and z_1 dominates z_2. Let $j \in [0, d - 1]$ be the minimum index such that $z_1 = y_j$.

- If $z_1 = x$, then $z_2 = y_i$ for some $i \in [1, d - 1]$. By Lemma 1 we conclude that in iteration i, $(x+1)$ mod $b^i = 0$. Therefore, $z_2 = n_i$ is not added to σ, which is a contradiction for z_2 being in σ.
- If $z_1 \neq x$, then $z_1 = y_j$ and $z_2 = y_i$ for some $1 \leq j < i \leq d-1$. By construction, the digits $y_{i,d-i+1} \cdots y_{i,d}$ are equal to $b - 1$. Since y_j dominates y_i, this also

means that the digits $y_{j,d-i+1} \cdots y_{j,d}$ are equal to $b-1$. Now, note that y_{j-1} is greater than y_j, and they can differ only in indices $d-j$ and $d-j+1$. Additionally, $d-i+1 \geq d-j$ (since $j < i$) and so $y_{j,d-j}$ and $y_{j,d-j}$ are equal to $b-1$. We conclude that $y_{j-1,d-j}$ and $y_{j-1,d-j}$ are also equal to $b-1$. That is, $y_{j-1} = y_j$ which contradicts the minimality of the index j.

In both cases, we reach a contradiction, which completes the proof. □

Now, we are ready to prove that σ is indeed a dominating partition for base b and integer x.

Lemma 3. *σ is a dominating partition for base b and integer x.*

Proof. By construction, the maximum element of σ is x. It remains to show that for every integer $z \in [0, x-1]$, there exists some index i such that $y_i \leftarrow \left\lfloor x/b^i \right\rfloor \cdot b^i - 1$ such that y_i dominates z.

Let k be the minimum index for which $z_k > x_k$ and let $j \in [1, k-1]$ be the maximum index such that $z_j < x_j$. We start by proving that z is dominated by $y_{d-j} = \left\lfloor \frac{x}{b^{d-j}} \right\rfloor \cdot b^{d-j} - 1$. It holds that the digits $y_{d-j,j+1} \cdots y_{d-j,d}$ are equal to $b-1$, proving that $y_\ell \geq z_l$ for each $\ell \in [j+1, d]$. Since the index k is minimal, and $j < i$, $z_1 \cdots z_j$ is dominated by $x_1 \cdots x_j$. Together with the fact that $y_{d-j,t} = x_t$ for each $t \in [1, j-1]$, we get $y_{d-j,t} \geq z_k$ for those indices. Finally, $y_{d-j,j} = x_j - 1$ and since $z_{d-j,j} < x_j$ it follows $y_{d-j,j} \geq z_j$.

Therefore, since y_{d-j} is added to σ in the $(d-j)^{\text{th}}$ iteration of Algorithm 1, the claim follows. □

Finally, we show that σ is a minimum dominating partition for base b and integer x.

Theorem 1. *For any two positive integers x and b, the sequence output by $\mathsf{MDP}_b(x)$ defined in Algorithm 1 is a minimum dominating partition for x in base b.*

Proof. By Lemma 3, $\mathsf{MDP}_b(x)$ is a dominating partition for base b and integer x. To prove that it is indeed of minimum size, suppose there is a smaller dominating partition $\mathsf{OPT}_b(x)$, where $|\mathsf{OPT}_b(x)| < |\mathsf{MDP}_b(x)|$. By the pigeonhole principle, there is some element $y \in \mathsf{OPT}_b(x)$ that dominates two distinct elements $z, w \in \mathsf{MDP}_b(x)$. There are two cases to consider:

- Case 1: z dominates y. Then, since y also dominates w, by transitivity we have that z dominates w, which contradicts Lemma 2.
- Case 2: z does not dominate y. Then there must be some other $v \in \mathsf{MDP}_b(x)$ which does dominate y, and again by transitivity we have that v dominates z, which contradicts Lemma 2.

In either case, we arrive at a contradiction, which proves the claim. □

2.5 Efficient Hashchains

Let H be a collision-resistant hash function. For a positive integer k, recall that the commitment of a hash chain of k with uniformly sampled randomness r is defined as $\mathsf{H}^k(r)$. For an integer t, a prover can supply the value $\pi = \mathsf{H}^{k-t}(r)$ to a verifier, which will be accepted so long as $k \geq t$.

Let N be an integer and let b be a base, setting $d = \lceil \log_b(N) \rceil$, and let $k_1 \cdots k_d = k$ represent the base-b decomposition of k. For uniformly sampled randomness $r = \langle r_1, \ldots, r_d \rangle$, we define the commitment of a *hash multichain* of length d, denoted $\mathsf{HMC}_d(k; r)$, as the following concatenation of values:

$$\mathsf{HMC}_d(k; \langle r_1, \ldots, r_d \rangle) \coloneqq \mathsf{H}(\mathsf{H}^{k_1}(r_1) \mid\mid \cdots \mid\mid \mathsf{H}^{k_d}(r_d)).$$

Analogously, for an integer $t = t_1 \cdots t_d$, a prover can supply the sequence of value $\pi = \mathsf{H}^{k_1 - t_1}(r_1) \mid\mid \cdots \mid\mid \mathsf{H}^{k_d - t_d}(r_d)$, which proves that $t \in [0, k]$ as long as k *dominates* t, since otherwise $k_i - t_i$ is not positive for all $1 \leq i \leq d$.

HashWires subroutine. For an integer d, we define a randomized procedure HWS_d which takes as input an integer k and outputs a permutation $P : [d] \to [d]$ along with a pair of sequences each of d elements of $\{0,1\}^\lambda$. For randomness $r \in \{0,1\}^\lambda$, the procedure $\mathsf{HWS}_d(k; r)$ operates as follows:

1. For each $i, j \in [d]$, the subroutine computes $r_{i,j} \leftarrow \mathsf{H}((i,j) \mid\mid r)$. It also uses r and H to derive a pseudorandom permutation $P : [d] \to [d]$.[2]
2. The subroutine computes $\langle w_1, \ldots, w_m \rangle \leftarrow \mathsf{MDP}_b(k)$. For each $i \in [m]$, the subroutine computes $a_i \leftarrow \mathsf{HMC}_d(w_i, \langle r_{i,1}, \ldots, r_{i,d} \rangle)$, and then samples $d - m$ random values $a_{m+1}, \ldots, a_d \leftarrow_{\mathsf{R}} \{0,1\}^\lambda$.
3. For each $i \in [d]$, define $c_i \leftarrow a_{P(i)}$. The output is $(P, \langle c_1, \ldots, c_d \rangle, \langle r_{1,1}, \ldots, r_{d,d} \rangle)$.

2 We omit the details for how this permutation is sampled, with the only requirement that the evaluations of H on r are done in a domain-separated manner (so as to not interfere with other hash evaluations in the protocol).

Note that the length of the hashchains implicitly defined by $\mathsf{HWS}_d(k;r)$ are *logarithmic* in k (linear in its representation), whereas applying the PayWord hashchain construction here would instead yield a hashchain *linear* in k (exponential in its representation). In the next section, we will show how to take advantage of this exponential reduction in hash evaluations in order to construct an efficient CBRP protocol.

3 The HashWires Construction

In this section, we present our main result: HashWires, a construction of a credential-based range proof that relies only on the existence of collision-resistant hash functions. HashWires works by constructing a commitment from a collection of hash multichains derived from a minimum dominating partition (Definition 9) of an integer x. These sequences of hash multichains are arranged in a structure in which, similar to the original PayWord construction, intermediate nodes of the chain are included to form the range proof.

Let N be a positive integer and let b be a base. Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a random-output hash function. Let $\mathsf{ACC} = (\mathsf{ACC.Setup}, \mathsf{ACC.Eval}, \mathsf{ACC.WitCreate}, \mathsf{ACC.Verify})$ be a secure accumulator. The HashWires construction $\mathsf{HW} = (\mathsf{HW.Setup}, \mathsf{HW.Commit}, \mathsf{HW.Prove}, \mathsf{HW.Verify})$ is defined as follows:

- $\mathsf{HW.Setup}(1^\lambda) \to \mathsf{pp}$. On input the security parameter λ, the setup algorithm computes $\mathsf{pp}_{\mathsf{ACC}} \leftarrow \mathsf{ACC.Setup}(1^\lambda)$, $d = \lceil \log_b(N) \rceil$, and outputs $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ACC}}, N, b, d)$.
- $\mathsf{HW.Commit}(\mathsf{pp}, x; r) \to \mathsf{com}$. On input the public parameters $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ACC}}, N, b, d)$, an integer $x \in [N]$, and randomness r, the commit algorithm computes $(P, \langle c_1, \ldots, c_d \rangle, \langle r_{1,1}, \ldots, r_{d,d} \rangle) \leftarrow \mathsf{HWS}_d(x; r)$ and outputs $\mathsf{com} \leftarrow \mathsf{ACC.Eval}(\mathsf{pp}_{\mathsf{ACC}}, \{c_1, \ldots, c_d\})$.
- $\mathsf{HW.Prove}(\mathsf{pp}, x, t; r) \to \pi$. On input the public parameters $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ACC}}, N, b, d)$, an integer $x \in [N]$, a threshold integer $t \in [N]$, and randomness $r \in \{0,1\}^\lambda$, the prove algorithm computes $(P, \langle c_1, \ldots, c_d \rangle, \langle r_{1,1}, \ldots, r_{d,d} \rangle) \leftarrow \mathsf{HWS}_d(x; r)$. Let $\langle w_1, \ldots, w_m \rangle \leftarrow \mathsf{MDP}_b(x)$, and let $i^* \in [m]$ be the first index for which w_{i^*} dominates t. Denoting $y_1 \cdots y_d$ as the base-b representation of w_{i^*} and $t_1 \cdots t_d$ as the base-b representation of t, for each $j \in [d]$, define $v_j \leftarrow \mathsf{H}^{y_j - t_j}(r_{i^*,j})$. The prover algorithm computes

  $$\mathsf{wit} \leftarrow \mathsf{ACC.WitCreate}(\mathsf{pp}_{\mathsf{ACC}}, c_{P(i^*)}, \{c_1, \ldots, c_d\})$$

  and outputs the proof $\pi \leftarrow (\mathsf{wit}, \langle v_1, \ldots, v_d \rangle)$.
- $\mathsf{HW.Verify}(\mathsf{pp}, \mathsf{com}, t, \pi) \to z$. On input the public parameters $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ACC}}, N, b, d)$, a commitment com, a threshold $t \in [N]$ with $t_1 \cdots t_d$ as its base-b representation, and a proof $\pi = (\mathsf{wit}, \langle v_1, \ldots, v_d \rangle)$, the verify algorithm computes

  $$c^* \leftarrow \mathsf{H}\left(\mathsf{H}^{t_1}(v_1) \,||\, \cdots \,||\, \mathsf{H}^{t_d}(v_d)\right).$$

  The verify algorithm outputs the bit $z \leftarrow \mathsf{ACC.Verify}(\mathsf{pp}_{\mathsf{ACC}}, \mathsf{com}, \mathsf{wit}, c^*)$.

3.1 Security

In order to prove that HashWires is indeed a secure credential-based range proof, we must show that the commitment is binding and that the range proof satisfies completeness, commitment-conditional soundness, and witness indistingishability[3]. We provide an intuition behind the proofs of each of these properties below.

Security intuition. Informally, the binding property of the commitment, as well as the perfect completeness of the range proof are mirrored from the constructions from PayWord and the use of hash chains, as they also apply in HashWires.

Commitment-conditional soundness comes from the property, ensured by the minimum dominating partition construction, that for any $y > x$, there is no element of $\mathsf{MDP}_b(x)$ which dominates y. This effectively implies that in order for a dishonest prover to fool the verifier into accepting, the prover must compute a pre-image for the hash function H corresponding to the digit in which an element of $\mathsf{MDP}_b(x)$ does not dominate y (thereby violating the collision-resistance property of H).

Finally, for witness indistinguishability, we leverage the simulation-independent property of each of the individual hash multichains corresponding to each element of $\mathsf{MDP}_b(x)$. It suffices to then show that our mechanism for *combining* the roots of these hash multichains still preserves the witness indistinguishability across the multichains. Here, we rely on the fact that the prover supplies dummy entries to pad the number of branches to be consistent and independent of the size of $\mathsf{MDP}_b(x)$, and then permutes the branches so that the verifier cannot tell which element of $\mathsf{MDP}_b(x)$ dominates x.

3 As previously noted, these last two properties are weaker notions than what is typically required for ZKRPs (soundness and zero knowledge).

The proof of the following theorem is given in Appendix A.

Theorem 2. *The construction* HW *is a secure CBRP, based on the assumption that* H *is a collision resistant and random-output secure hash function, and that* ACC *is a secure accumulator.*

4 Experimental Results

To assess the practicality of HashWires, we provide an open-source Rust implementation [23] of the construction, measuring its performance on a range of parameters. We describe a number of implementation optimizations that we integrated on top of the straightforward version of HashWires presented in Section 3 in order to achieve better performance. We conclude with a comparison against other ZKRPs in terms of overall efficiency.

Instantiating primitives. The two main primitives that need to be configured for HashWires are the underlying hash function (which must satisfy collision resistance and random-output security) and the secure accumulator. We outline these choices below:

- Hash function: Our implementation can be configured to any hash function which implements the digest trait in Rust, which includes Blake2, Blake3, SHA256, SHA512, and others.
- Accumulator: We use a sparse Merkle tree implementation that is also parametric in the choice of underlying hash function.

HashWires also requires an instantiation of a random permutation, for which we rely on a variant of Fisher-Yates algorithm for shuffling a sequence of elements.

For handling large integers, we use the num_bigint Rust library in order to convert from a string representation of a large integer, along with an appropriate zero-padding. The algorithm we use for computing minimum dominating partitions matches Algorithm 1.

4.1 Optimizations

The most straightforward optimization of HashWires is the configuration of the base representation, which introduces a natural tradeoff between proving and verification time versus proof size. In the extreme example, when $b = N$, HashWires is equivalent to PayWord, with constant-size proofs but exponential proving and verification time complexity. On the other hand, setting $b = 2$ yields a highly parallelizable set of operations for the prover and verifier, at the cost of larger (but still $O(n)$-sized) proofs.

There is also value in choosing a base that lies at neither end of these extremes. For instance, if $n = 64$ (imagine the commitments are of Unix timestamps), we can achieve relatively short proofs while paying a cost in prover and verification time, but since hash computations are orders of magnitude faster than the group operations that existing range proof solutions use, HashWires can offer solutions for these instances which are more efficient both in proof size *and* prover and verification time. In Section 4.3, we show for a variety of common settings for n, how a base can be chosen in order to achieve smaller proof sizes than directly applying Bulletproofs.

4.1.1 Truncation and Padding

The basic HashWires scheme does not take truncation and padding into consideration. However, as we will show, a careful selection of accumulators can result to faster operations and significantly shorter proofs without sacrificing security. There are two places where such optimizations can be applied.

First, we can omit the leading zeros of any number represented in base b up to the maximum N. For instance, imagine the issuing number $k = 47$ in base $b = 10$ and $N = 999,999$. This number is encoded as $0*b^5+0*b^4+0*b^3+0*b^2+4*b^1+7*b^0$ and would typically require 6 independent *hash chains*, one per base-10 digit. However, by omitting the leading zeros we would require two chains only. It is highlighted though that whatever the truncation logic, the proof should not reveal the size of the number (degree of polynomial), and thus a special accumulator should be selected to hide the number of multichains. After considering various accumulators to optimize on proof size, we recommend PLA, a special hash-based accumulator described below.

Similarly, another opportunity to apply padding is to hide the number of MDP-values. In the above example, the MDP-list for $k = 47$ consists of two elements $\langle 47, 39\rangle$ in base-10. Ideally, only these two MDP values should be included in computations, but without exposing the size of this list. This is possible via padded deterministic sparse Merkle trees, which efficiently allow for hiding the MDP-list size and at the same time,

requiring as many hash multichains as the number of MDP elements.

Padded Linear Accumulator (PLA). We describe PLA, a simple linear accumulator which, by taking advantage of ordered-sets and padding opportunities, offers a single hash-output element inclusion proof when used in HashWires. As mentioned already, our protocol requires hash multichains, where each chain i corresponds to the i-th digit of a number k in base b. In practice, it is very common that the issuing and threshold values k and t are by far smaller than N, and thus, by skipping the leading zeros, less b-base digits (and inherently number of chains) than $d_N = \lceil \log_b(N) \rceil$ are required in the computations.

PLA takes advantage of the above observation and is best suited to applications where:

- insertion order of the elements matters;
- we prove inclusion of the last m elements in the set;
- proofs must not leak the set's size.

The above properties fit exactly the requirements of HashWires, because (a) hashchains are ordered per significant-digit (b) it requires inclusion proof of adjacent chains and (c) the number of actual chains used in the commitment should not be leaked as this would reveal information about the size of the issued value.

Let N be the maximum supported integer and k the issuing value and let b be a base, setting $d_k = \lceil \log_b(k) \rceil$, and let $k_1 \cdots k_{d_k} = k$ represent the base-b decomposition of k, where k_1 is the most significant digit. For uniformly sampled randomness r, we define the accumulator's Eval function of a *hash multichain* of length d_k, denoted $\mathsf{PLA}_d(k; r)$, as follows:
$\mathsf{PLA}_d(k; \langle r_1, \ldots, r_d \rangle) = z_d$, where z_d is computed differently depending on the padding requirement. If $d_k = d_N$ then padding does not need to apply because all of the possible digits/chains are used anyway and $z_d = \{z_0 \leftarrow \mathsf{H}^{k_1}(r_1), z_i \leftarrow (z_{i-1} \mid\mid \mathsf{H}^{k_{i+1}}(r_{i+1}))\}$ for $i \in [1, d_k)$. Otherwise, a padding element p is deterministically generated from the issuing secret key and $z_{d_k} = \{z_0 \leftarrow p, z_i \leftarrow (z_{i-1} \mid\mid \mathsf{H}^{k_i}(r_i))\}$ for $i \in [1, d_k]$.

As for PLA's WitCreate inclusion proof, the prover outputs the result of the above Eval function up to $d_k - d_t$, where t is the requested range proof threshold decomposed as $t = t_1 \cdots t_{d_t}$ and d_t its number of b-base digits, computed as $d_t = \lceil \log_b(t) \rceil$. Then, the verifier can "continue" evaluating the received output of Eval with the computed hashchain elements $\mathsf{H}^{k_i - t_i}(r_i)$. We highlight that because the verifier only receives the output of Eval for the first $d_k - d_t$ elements, d_k itself is not exposed and thus the number of actual chains stays secret; otherwise, that would leak information about the issued value.[4]

To sum up, the main advantage of PLA compared to a Merkle-tree accumulator is that the inclusion proof is independent of N and it consists of at most one hash output element versus the expected d_N if we used Merkle-tree accumulators. Additionally, as already mentioned in sections 2.5 and 3, the basic HashWires construction computes the commitment of each MDP's hash multichain using a simple *hash concatenation* accumulator $\mathsf{HMC}_d(k; r) \coloneqq \mathsf{H}(\mathsf{H}^{k_1}(r_1) \mid\mid \cdots \mid\mid \mathsf{H}^{k_d}(r_d))$. However this does not take padding into consideration and without PLA we have to include $d_N - d_k$ hash elements to hide the number of chains used to encode k. The latter would result to significantly longer proofs, especially when $t << N$. To avoid this, we propose updating HMC_d to $\mathsf{PLA.Eval}(\mathsf{pp}_{\mathsf{PLA}}, \{\mathsf{H}^{k_1}(r_1), \ldots, \mathsf{H}^{k_d}(r_d)\})$.

4.1.2 Deterministic Sparse Merkle Trees

As already mentioned in Section 3, the final step of a HashWires commitment requires an accumulator output over each MDP's multichain commitment c_i and the basic construction computes this as $\mathsf{com} \leftarrow \mathsf{ACC.Eval}(\mathsf{pp}_{\mathsf{ACC}}, \{c_1, \ldots, c_d\})$.

However, just naively concatenating all c_i has two major efficiency drawbacks. Firstly, it is highlighted that when a proof is requested, only one MDP path is actually selected. Unfortunately, this linear concatenation accumulator expects attaching all c_i values to the proof, otherwise a verifier cannot evaluate the accumulator's outcome; this results to longer proofs. Secondly, apart from compressing the final HashWires commitment into one element, the purpose of this accumulator is to also hide the actual MDP(k)-list size, $d_k \leftarrow \lceil \log_b(k) \rceil$. Obviously, this is because d_k leaks information about the value k. It has been shown that $d_k \leq d_N$ and thus one option to hide d_k is to pad c_i values with with $d_N - d_k$ random *fake* elements. In some scenarios, i.e., when a small base is used and our issued number is very small, adding extra $d_N - d_k$ elements might significantly increase computational effort and proof size.

One option to circumvent the first issue is replacing the *concatenation* accumulator with a Merkle tree. However, this tree should have d_N (and not d_k) leaves

4 Obviously, if $d_k - d_t = 0$, then the output *wit* of WitCreate is empty and extra inclusion proof elements are not required.

as the tree-height would disclose information about the number of its leaves, and inherently the MDP(k)-size. That said, indeed a naive Merkle-tree requires $\log_2(d_N)$ inclusion proof, but it still requires computing $d_N - d_k$ random *fake* leaves. Fortunately, we can reduce the number of extra padding nodes by utilizing the deterministic padded sparse Merkle tree (SMTree) approach of the DAPOL [15] protocol, which still allows for a d_N tree-height, but requiring a logarithmic number of padding nodes. We implemented and applied the SMTree accumulator as the final step of the optimized HashWires construction.

4.2 Reusing Multichains

In the basic version of HashWires each MDP value has its own independent *multichain* and thus, in the worst-case $\log_b^2(N)$ hashchains of length b are required. The latter is because there exists a maximum of $\log_b(N)$ MDP values and each MDP integer is encoded with a maximum of $\log_b(N)$ hashchains. Fig. 2 depicts a simple example of such a case.

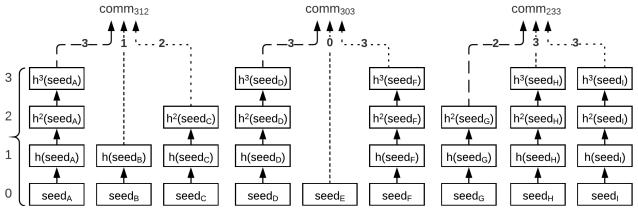

Fig. 2. Basic base-4 HashWires commitment to number 303 using 9 variable-size hashchains, 3 per MDP value $\langle 312, 303, 233 \rangle$.

However, we can take advantage of the fact that plain HashWires is a one-time proof and thus we could actually share multichains between MDP values by wisely wiring hashchain nodes. Briefly, one can imagine this optimization technique as overlaying the multichain groups of Fig. 2 and merging them into one single multichain as shown in Fig. 3. As a result, only $\log_b(N)$ hashchains are required which reduces the commitment and proof generation costs by a factor of $\log_b(N)$, which for base-256 implies an almost eight times faster HashWires construction[5].

Sharing the multichain group does not come for free though. As mentioned already, the size of the MDP-list should remain confidential because that would leak information about the structure of the issued value. As we show below, without extra modifications to the protocol, given a HashWires proof for a requested range r, a verifier can try various hashchain size combinations out of the received proof values and reach to other than the intended MDP commitment.

To visualize how this is possible we will use Fig. 3 as an example use case, which shows an $\mathsf{MDP}_4(312)$ instance. Assuming a less than or equal to 301 proof is requested, the prover normally picks the $comm_{303}$ path (because 303 is smallest MDP-value larger than 301). Thus, the proof should contain the following list of nodes: $L = [seed_A, seed_B, h^2(seed_C)]$. An honest verifier would hash $seed_A$ three times, would use $seed_B$ directly, and hash $h^2(seed_C)$ once to produce the wires required to compute $comm_{303}$.

However, a careful reader will realize that malicious verifiers can brute force the number of rehashes for different combinations until they reach another MDP-value. In our example, if one hashes $seed_A$ twice, $seed_B$ three times and $h^2(seed_C)$ once, an adversary will reach to $comm_{233}$. Similarly, with different combinations one can reach to $comm_{312}$. As a result, the full size of L is leaked.

To defend against the above scenario, we introduce an extra salt value, in order to obfuscate each MDP commitment root before they get used as inputs to the final HashWires accumulator. That said, each proof should be accompanied by an extra salt value (typically 16 bytes long should be enough) and the final accumulator during HashWires commitment generation should take as input $\mathsf{com} \leftarrow \mathsf{ACC.Eval}(\mathsf{pp}_{\mathsf{ACC}}, \{PRF(c_1, salt_1), \ldots, PRF(c_d, salt_d)\})$, where an HMAC or Key Derivation Function (KDF) can be used as a PRF candidate. Due to the existence of this salt, the above attack cannot apply because the proof includes the salt for the intended MDP-value only, and thus, an adversary cannot compute/predict the PRF output of any other brute-forced MDP path required to evaluate the final accumulator. Fig. 6 in Appendix B shows an example of adding salts to each MDP commitment before used as an input to the final accumulator.

5 Note that HashWires was named after the hash-node wiring technique that allowed multichain sharing between MDP values, which boosted performance by a significant constant factor.

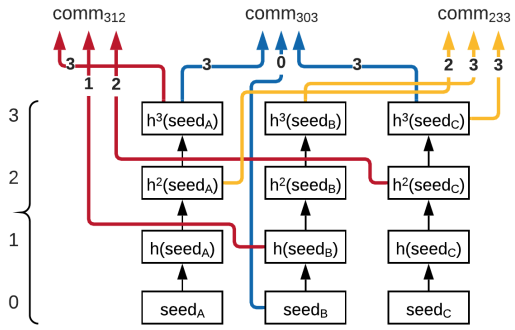

Fig. 3. Optimized base-4 HashWires commitment to number 303 by reusing *multichains*.

4.3 Benchmarks and Comparisons

Recall that we use b to represent the base parameter for HashWires, N to denote the maximum integer of the domain $[0, N]$ for which our range proof intervals lie. Asymptotically, HashWires benefits from:

- $O(1)$ commitment size,
- $O(\log_b(N))$ proof size, and
- $O(\log_b(N))$ proof generation and verification time.

Note that although PayWord has $O(1)$ commitment size and $O(1)$ proof size, it suffers from $O(N)$ proof generation and verification time, which is completely impractical for intervals represented by a 64-bit integer (or larger).

In the following, we give further justification to the practicality of HashWires by concretely analyzing its efficiency.

4.3.1 Comparing Proof Size

To generate a HashWires commitment efficiently, a maximum of $\log_b(N)$ hashchains of size b are required, along with $\log_b^2(N)$ total wires (see Fig. 3). Then, we use a Merkle tree accumulator to create a root for the shuffled sub-commitments of each MDP value (see Fig. 6).

In the worst case, where padding is not applied, the total size for a single HashWires range proof is $\log_b(N) + \log_2(\log_b(N))$ hash elements, which for $n \leq 128$, depending on the base selection, compares favorably against a naive solution that uses Bulletproofs outputting proofs of $2\log_2(\log_2(N)) + 9$ elliptic curve (EC) elements over a Pedersen commitment (typically, each compressed EC element is 32 bytes, equal to a 256-bit hash output). Note that the final optimized HashWires version requires an extra 16 bytes for the MDP-salt plus 1 byte to encode the leaf-index required for the final inclusion path to the sparse Merkle tree accumulator. See Table 1 for our results. Note that size comparisons also include the trusted-setup based Groth16 [28] (over BLS12-381 curve) constant proof size zero knowledge proof system. This is added for reference purposes only, as the security of HashWires is based on a weaker model.

n	BPs	G16	HW_4	HW_{16}	HW_{256}	HW_{2048}
16	544	192	369	209	113	113
32	608	192	657	369	209	177
64	692	192	1201	657	369	305
128	736	192	2257	1201	657	529

Table 1. Range proof sizes (in bytes). We use HW_b to denote HashWires over a base b, with the security parameter $\lambda = 128$. We use BPs to denote Bulletproofs and G16 for Groth16 [28].

Finally, we note that these numbers were obtained under the assumption that the security parameter is 128 bits (which results in 256-bit outputs for the hash function when relying on collision resistance). Further improvements to proof size can be leveraged by building HashWires out of pseudorandom functions, which could potentially allow us to rely on smaller overall output sizes for each component of the proof.

4.3.2 Computational Efficiency

We implemented in Rust the optimized HashWires scheme by applying PLA and sparse Merkle Tree accumulators and reusing multichains between MDPs [23]. We then compared our base-16 and base-256 worst case results against the faster available Rust Bulletproofs implementation from the dalek-cryptography crate. We also ran Bulletproofs using the AVX2 backend, which implements curve arithmetic using parallel formulas via vectorized instructions [40] to ensure efficiency transparency against optimized range proofs implementations.

Table 2 presents the cost in microseconds (lower is better) when running benchmarks in a MacBook Pro 2.4 GHz 8-Core Intel Core i9 CPU with 32GB or RAM. As shown, Blake3 performs better than SHA2 in our selected Intel CPU, although it's true that in AMD Ryzen architectures that support SHA extensions BLAKE3 is only slightly faster than hardware-accelerated SHA-256. Another interesting result is that because proof generation internally regenerates the commitment to com-

pute the final tree-inclusion proofs, it is slightly more expensive than commitment creation. To sum up, when comparing Blake-3 backed HashWires in base-16 against the AVX2 optimized BulletProofs, the former performs about 60 times better on proof generation and 30 times on proof verification. Obviously, a Bulletproofs commitment is just a Pedersen compressed elliptic curve point which is computed 3 times faster than the best HashWires mode in the table[6].

scheme	$\text{commit}_{\text{gen}}$	$\text{proof}_{\text{gen}}$	$\text{proof}_{\text{ver}}$
BPs	71	12099	1555
BPs AVX2	36	6516	938
HW_{16} SHA2-256	274	278	84
HW_{256} SHA2-256	651	656	619
HW_{16} Blake3	101	103	31
HW_{256} Blake3	260	263	230

Table 2. Efficiency comparison (in microseconds) between Bulletproofs (BPs) with and without AVX2 support, and HashWires for different bases and hash functions.

5 Extensions and Applications

In this section, we describe a number of extensions to the core HashWires construction which may be relevant for various applications. We conclude with a high-level description of potential applications for CBRPs.

5.1 Handling Intervals

Recall that in our definition of a CBRP, the prover need only show that the committed integer is above a certain threshold, whereas typically range proofs are designed to show that the committed integer lies within an interval (both an upper and lower bound).

A fairly straightforward method for extending HashWires to handle interval range proofs is to produce two separate commitments of a value x, one with a range proof that $x \geq t$ for some threshold t, and then another proof that $N - x \geq N - t$, as described in [2]. Since both commitments are simply hash outputs, they can be combined with another hash operation to produce a "root" commitment that the verifier can check both range proofs against. Recall the left half of Fig. 4 for an illustration of this method.

Note that we critically rely on the issuer being able to verify that the commitment is well-formed, and hence why it must behave as a trusted third party—otherwise, it would be possible for a prover to generate range proofs that appear valid to a verifier, yet do not correspond to a well-formed commitment.

Conjunctions. Even more generally, what we are showing is that HashWires commitments are amenable to supporting conjunctions. This means that we can apply them to combine multiple proofs across a *set* of secret integers and their commitments together, rather than just a single integer. In terms of applications, this can be used in order to prove the inclusion of, say, a two-dimensional point in a given area of a surface; see the right half of Fig. 4 for such an example, where two interval sub-proofs, one for longitude and another for latitude can just be combined into one single HashWires proof.

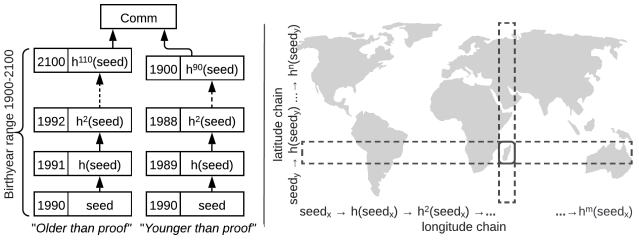

Fig. 4. Left: older/younger than interval proofs (born in 1990). Right: 2D location range proofs (somewhere in Madagascar).

5.2 Reusable Proofs

One inherent limitation of the HashWires construction is that witness indistinguishability is not guaranteed to be preserved if multiple range proofs must be generated *for the same commitment*. To support the "reusability" of the commitment in this manner, we can interpret the commitment as a leaf of a balanced binary Merkle tree, whose root commitment acts as the reusable commitment for the protocol. With T leaves in this Merkle tree, the new protocol would be able to support T distinct range proofs over the same commitment, while only incurring an additive $O(\log T)$ factor in proof size and verification time. However, commitment and proof generation time is multiplied by $O(T)$. Based on the results presented in Table 2, a 60-times HashWires (base-16 w/ Blake3) would still be faster than Bulletproofs proof

6 We highlight that Bulletproofs is a many-times range proof scheme, which implies that in the long run, a few-times HashWires base-16 construction performs better only if used to produce 30 proofs or less.

generation. See Fig. 5 for an illustration of this standard technique for a many-times stateful HashWires, similarly to that used in a Merkle Signature Scheme (MSS) [32].

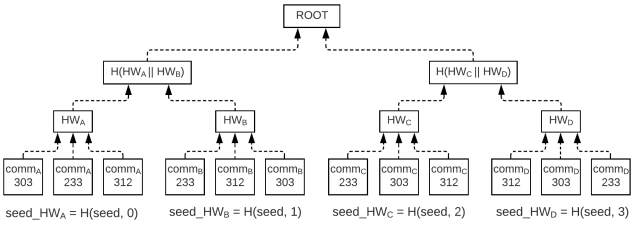

Fig. 5. Stateful 4-times Merkle-ized HashWires à la MSS.

5.3 Preventing Malleability

Inherent to the hashchain system, it is important to notice that our proofs are malleable: a proof of the value $311 < 312$ can be reused to prove that $300 < 312$ or that $211 < 312$, and so on for other values dominated by 311, which can be desirable or not.

For instance, malleability can be considered a feature when reusing the same proof for different requested ranges, without computing each one from scratch. As an example, full payslip copies are a common request and works as an important metric for landlords because it shows them that the potential renter has a steady income and will be able to make the monthly payments. For privacy purposes (not everyone feels confident on revealing their salary break down details), renters can just request a HashWires commitment from their employers, then compute a single range proof for the highest requested value and if possible reuse this to prove smaller ranges to other landlords. Similarly, real estate agents might request one single maximum range proof from renters (their customers); then brokers are able to recalibrate these proofs per landlord without renter-landlord interaction.

In other settings though it is considered an adaptive chosen message attack when a forged range proof proves a range other than the value that was legitimately generated; in these cases malleability should be avoided. For instance, the original PayWord protocol is malleable by default and when used for auction bids, a malicious man-in-the-middle can alter range proofs and submit a valid, smaller than the original, bid. Changing bids would result to auction manipulation favoring dishonest users.

There exist at least two approaches to defend against malleability. A straightforward solution is for the prover to always sign both the HashWires proof and the requested numeric range value. A signature scheme is binding to the message, and thus, that would work in most blockchain applications as they typically require an address-owner signature anyway.

We could however make HashWires self-protected against malleability without the need of extra signatures using ideas from hash-based post-quantum schemes. Actually, one can easily find similarities between HashWires and the Winternitz one-time signature scheme (WOTS) [9, 32]. In Layman terms, the hashchain part of HashWires can be considered a variation of WOTS, with issuer-restrictions on the byte-values that could be signed/proven. In short, while in WOTS any bit combination can be signed, in HashWires the issuer ensures that multichains are computed in such a way that not all values have a path to the commitment root. WOTS defends against adaptive chosen message attacks by signing over a checksum value appended to the (hashed) message bytes at the cost of extra hashchains. HashWires can be enhanced with exactly the same mechanism.

To understand how it works, imagine that each hashchain can only be computed in the forward direction due to the computational implausibility to find a hash pre-image. The checksum is computed as $C = \sum_{i=1}^{n}(b-1-M_i)$ and it is a value that consists of the sum of the differences between the $b-1$ and M_i, where M_i is the i^{th} base-b digit of the requested range value. Similarly to WOTS, the resulting checksum sum is encoded as a base-b integer and appended to the proof. For instance, in the 32-bit domain, the value $2^{32}-1$ is encoded as (255, 255, 255, 255) in base-256 and would have a checksum of 0. Similarly, the message (0, 0, 0, 1) would have an integer checksum of 1019, which is encoded as (3, 251) in base-256 [27]. In practice, when using base-256, the checksum adds up an extra two multichains, thus resulting to 64 bytes longer proofs for up to 8192-bit domains)[7].

Briefly, the WOTS checksum trick works because if an attacker tries to prove a smaller range, the checksum should be increased, but due to the one-way nature of hash functions, adversaries cannot produce longer chains for the checksum part. More information about

7 In theory, one can fit the checksum in one longer single chain of (worst-case) length $\lceil log_b(N) \rceil * b$, instead of breaking it to b-size multichains as in WOTS. Compared to the malleable HashWires mode, that reduces efficiency by a factor of 2 at most.

the security of the Winternitz checksum can be found in [9].

5.4 Applications

We briefly outline a set of applications for HashWires (and CBRPs in general) in various practical settings, especially in constrained environments such as when smart-cards or IoT sensors are utilized or when algebraic group primitives are either not available or very complex to be securely implemented (i.e., hardware, embedded software or programming languages with no native support for ZKP systems). It is highlighted though that HashWires commitments are not homomorphic and thus cannot be used for adding confidential amounts.

Verified location. CBRPs can be extended to prove that multi-dimensional data verifiably lies within an interval range (such as 2D / 3D location) by issuing a proof for combined longitude and latitude. There exist many sensitive location-sensitive applications, including mobile and web user tracking (with privacy) *and* satellite TV and radio station receiver location/area restrictions. Many fog computing applications in particular rely on location awareness including actuators networks and wireless IoT sensors [45]. There exist various cryptographic primitives whose security relies on verifiable location including private location proximity [34], position-based key exchange and multiparty computations [16], location-based identity keys [14] and access control [46] among the others. One of the most important aspects though is malicious actors provisioning fake locations, and GPS coordinates reported by users may not be reliable. To mitigate this problem, the notion of location verification which focuses on methodologies for securely verifying provided position-data, was originally introduced in [7] and further explored in [13, 16, 38, 39, 42, 45, 47]. Briefly, CBRP commitments could be issued by IoT positioning sensors (i.e., by embedded secure enclaves) and then it is up to the application logic to decide what range proof to generate.

Know Your Customer. The broader application space based on these specific examples includes KYC (Know Your Customer) credentials. Regulators have mandated financial institutions and networks to implement KYC validation, which could be used to perturb client location and provide privacy for age with the use of CBRPs. They are also widely applicable to other aspects of financial services, where they can be used to represent account balance, deposits, and income that can be attested to by trusted third parties such as banks. This would be useful for mortgage, credit card and loan applications as ownership of assets can be proved to a third party (verifier) without revealing private information. As already mentioned, one can imagine using a proof to a landlord when submitting a rental application without having to reveal personal wealth or income information.

Micropayments and auctions. PayWord [37] is a credit-based micro-payment protocol which involves a customer, vendor and broker and uses hashschains for better computational performance (it is in fact a HashWires instance for $b = N$). In short, imagine issuing bank-checks, where one can gradually spend funds until the full amount is exhausted; sellers should not necessarily learn the remaining balance. This fits perfectly to a blockchain application of digital remittances where account holders issue checks off-chain. However, although micro-payment systems focus on small values, today's tiny denominations in many cryptocurrencies and due to extreme exchange rate volatility, that might result to long hashchains in the original PayWord scheme, even when issuing checks of a few dollars value. We believe that because HashWires is efficient for large domains and many blockchains allow hash function smart contract instructions, it might be a great candidate for blockchain-based gradually redeemable checks.

Along the same lines, PayWord variants have been proposed for privacy preserving auctions [2], where for example the winner might not publicly reveal the bid offer, but only prove that the value is greater than (or less than depending on the auction type) the rest of the bids. In Vickrey sealed-bid auctions for example [41], the range of the winning bid might even stay secret from everyone as the final price paid is the second-highest bid price. HashWires might be a candidate protocol in these settings as well.

Timestamping. Outside of financial services, CBRPs can also be applied to achieve timestamp privacy for certificates issued by a certificate authority (CA). Using CBRPs, an entity that has received a certificate from a CA can prove to a third party that a certificate has not yet expired, without revealing the expiration date. Other metadata such as time of certificate issuance or IP address can also be wrapped in a range proof. In general, CBRPs would make certificates more privacy-preserving for clients without foregoing server-side verification.

Other applications. Other potential use cases might include proving ranking ranges in job resumes/CVs and

online game leader-boards (i.e., top 10 GPA) without disclosing the actual ranking-position. In some settings this might be desirable to avoid review biases when the job details demand that only a threshold should be surpassed. Similarly, food and drug producers which might need to satisfy some ingredient upper/lower values to get licensed, could in theory be issued signed CBRP commitments from Food and Drug Administration (FDA) or similar agencies and then just print/show a verifiable proof that the legal limits are satisfied without disclosing the full "recipes" of their products.

6 Acknowledgements

We would like to thank our shepherd Alfredo Rial and other anonymous reviewers for their valuable inputs. This research received no specific grant from any funding agency in the public, commercial, or not-for profit sectors.

References

[1] Masayuki Abe. A secure three-move blind signature scheme for polynomially many signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 136–151. Springer, 2001.

[2] Sebastian Angel and Michael Walfish. Verifiable auctions for online ad exchanges. In *ACM SIGCOMM*, 2013.

[3] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & Communications Security*, 2013.

[4] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. 2020.

[5] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT '00*, 2000.

[6] Stefan Brands. Untraceable off-line cash in wallet with observers. In *Annual international cryptology conference*, pages 302–318. Springer, 1993.

[7] Stefan Brands and David Chaum. Distance-bounding protocols. In *EUROCRYPT*, 1993.

[8] William J Buchanan. Zero-knowledge proof: Proving age with hash chains. Asecuritysite: https://asecuritysite.com/encryption/age, 2021.

[9] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the winternitz one-time signature scheme. In *AFRICACRYPT*, 2011.

[10] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE S&P*, 2018.

[11] Jan Camenisch, Rafik Chaabouni, et al. Efficient protocols for set membership and range proofs. In *ASIACRYPT*, 2008.

[12] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT*, 2001.

[13] Srdjan Čapkun, Mario Čagalj, and Mani Srivastava. Secure localization with hidden and mobile base stations. In *in Proceedings of IEEE INFOCOM*. Citeseer, 2006.

[14] Konstantinos Chalkias. Secure cryptographic protocols and applications based on bilinear pairings. 2010.

[15] Konstantinos Chalkias, Kevin Lewi, Payman Mohassel, and Valeria Nikolaenko. Distributed auditing proofs of liabilities. *ZKProof*, 2020.

[16] Nishanth Chandran, Vipul Goyal, Ryan Moriarty, and Rafail Ostrovsky. Position based cryptography. In *Annual International Cryptology Conference*, 2009.

[17] David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors. *CRYPTO'82*. Plenum Press, New York, USA, 1982.

[18] Gaby G Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In *ACM CCS*, 2015.

[19] Ivan Damgård. Practical and provably secure release of a secret and exchange of signatures. *J. Cryptol.*, 1995.

[20] Ankur Shah Delight. Zero knowledge proof of age using hash chains, 2017.

[21] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *CT-RSA*, 2015.

[22] Chris Dods, Nigel P Smart, and Martijn Stam. Hash based digital signature schemes. In *IMA International Conference on Cryptography and Coding*, pages 96–115. Springer, 2005.

[23] Novi Financial. Hashwires rust implementation. https://github.com/novifinancial/hashwires, 2021.

[24] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO '97*, volume 1294, pages 16–30. Springer, 1997.

[25] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[26] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. In *SIAM Journal on computing 18.1*, pages 186–208, 1989.

[27] Matthew Green. Winternitz checksum. https://blog.cryptographyengineering.com/winternitz-checksum, 2021.

[28] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT '16*. Springer, 2016.

[29] Hudson Jameson. Which cryptographic hash function does ethereum use? https://ethereum.stackexchange.com/questions/550, 2016.

[30] Iuon-Chang Lin, Min-Shiang Hwang, and Chin-Chen Chang. The general pay-word: A micro-payment scheme based on *n*-dimension one-way hash chain. *Des. Codes Cryptogr.*, 2005.

[31] Gregory Maxwell, 2016. https://people.xiph.org/~greg/confidential_values.txt.

[32] Ralph C Merkle. A certified digital signature. In *CRYPTO*, pages 218–238. Springer, 1989.

[33] Eduardo Morais, Tommy Koens, Cees van Wijk, and Aleksei Koren. A survey on zero knowledge range proofs and applications. *CoRR*, abs/1907.06381, 2019.

[34] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, Dan Boneh, et al. Location privacy via private proximity testing. In *NDSS*, 2011.

[35] Diem open-source contributors. Diem blockchain hash function support. https://github.com/diem/diem, 2021.
[36] Andrew Poelstra. Mimblewimble. 2016.
[37] Ronald L Rivest and Adi Shamir. Payword and micromint: Two simple micropayment schemes. In *International workshop on security protocols*. Springer, 1996.
[38] Naveen Sastry, Umesh Shankar, and David Wagner. Secure verification of location claims. In *Proceedings of the 2nd ACM workshop on Wireless security*, pages 1–10, 2003.
[39] Dave Singelee and Bart Preneel. Location verification using secure distance bounding protocols. In *Mobile Adhoc and Sensor Systems Conference*, 2005.
[40] Henry de Valence. Accelerating edwards curve arithmetic with parallel formulas. https://medium.com/@hdevalence/accelerating-edwards-curve-arithmetic-with-parallel-formulas-ac12cf5015be, 2018.
[41] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 1961.
[42] Adnan Vora and Mikhail Nesterenko. Secure location verification using radio broadcast. *IEEE Transactions on Dependable and Secure Computing*, 3(4):377–385, 2006.
[43] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.
[44] Ching-Nung Yang and Hsu-Tun Teng. An efficient method for finding minimum hash chain of multi-payword chains in micropayment. In *CEC '03*, pages 45–48. IEEE, 2003.
[45] Rupeng Yang, Qiuliang Xu, Man Ho Au, Zuoxia Yu, Hao Wang, and Lu Zhou. Position based cryptography with location privacy: A step for fog computing. *Future Generation Computer Systems*, 78:799–806, 2018.
[46] Mingwu Zhang and Tsuyoshi Takagi. Geoenc: Geometric area based keys and policies in functional encryption systems. In *ASICP*, 2011.
[47] Yanchao Zhang, Wei Liu, Yuguang Fang, and Dapeng Wu. Secure localization and authentication in ultra-wideband sensor networks. *IEEE J. Sel. Areas Commun*, 2006.

A Security Proof for HashWires

Each of the following lemmas independently establish the security properties that make up a CBRP, as defined in Section 2.1.

Lemma 4. *The construction* HW *satisfies perfect completeness.*

Proof. Since $x \geq t$, this means that an element of $\mathsf{MDP}_b(x)$ dominates t. Using $y_1 \cdots y_d$ as the base-b representation of this element and $t_1 \cdots t_d$ the base-b representation of t, perfect completeness follows directly from the proof construction and its definition of the values $v_1, \ldots, v_d$. Since $v_j = \mathsf{H}^{y_j - t_j}(r_{i^*,j})$, and the verifier takes these values and computes $c^* \leftarrow \mathsf{H}(\mathsf{H}^{t_1}(v_1) \,||\, \cdots \,||\, \mathsf{H}^{t_d}(v_d))$, it follows that the set $\{c_1, \ldots, c_d\}$ exactly matches the set for which $\mathsf{ACC.Eval}$ outputs com. Therefore, by the correctness of the accumulator, we have that $\mathsf{HW.Verify}(\mathsf{pp}, \mathsf{HW.Commit}(\mathsf{pp}, x; r), t, \mathsf{HW.Prove}(x, t; r)) = 1$. □

Lemma 5. *The construction* HW *satisfies the binding property for its commitments, based on the assumption that* H *is collision resistant and that* ACC *is a secure accumulator.*

Proof. Recalling the definition of a binding commitment, our goal is to show that for two distinct integers $x_0, x_1 \in [N]$, the probability that their commitments are identical is negligible. Consider the event where $\mathsf{HW.Commit}(\mathsf{pp}, x_0; r_0) = \mathsf{com} = \mathsf{HW.Commit}(\mathsf{pp}, x_1; r_1)$, which we want to show happens with negligible probability. For each $z \in \{0, 1\}$, let $(P^{(z)}, \langle c_1^{(z)}, \ldots, c_d^{(z)} \rangle, \langle r_{1,1}^{(z)}, \ldots, r_{d,d}^{(z)} \rangle) \leftarrow \mathsf{HWS}_d(x_z; r_z)$, and let $C^{(z)} = \{c_1^{(z)}, \ldots, c_d^{(z)}\}$. There are two cases to consider:

- Case 1: $C^{(0)} = C^{(1)}$. Recalling the definition of a minimum dominating partition, note that since $x_0 \neq x_1$, it follows that $\mathsf{MDP}_b(x_0) \neq \mathsf{MDP}_b(x_1)$. Since $C^{(z)}$ is constructed out of iterated hashes on r_z, where the number of iterations depends on $\mathsf{MDP}_b(x_z)$, this means that we can break the collision resistance of H.
- Case 2: $C^{(0)} \neq C^{(1)}$. Without loss of generality, let β be such that $\beta \in C^{(0)}$ and $\beta \notin C^{(1)}$. If the commitments of x_0 and x_1 are equal, then $\mathsf{ACC.Eval}(\mathsf{pp}_{\mathsf{ACC}}, C^{(0)}) = \mathsf{com} = \mathsf{ACC.Eval}(\mathsf{pp}_{\mathsf{ACC}}, C^{(1)})$, which allows us to construct an adversary $\mathcal{A}$ that participates in $\mathsf{Expt}^{\mathsf{ACC}}$ as follows:
  - The challenger sends $\mathsf{pp}_{\mathsf{ACC}} \leftarrow \mathsf{ACC.Setup}(1^\lambda)$ to $\mathcal{A}$.
  - $\mathcal{A}$ sends a witness $\mathsf{wit} \leftarrow \mathsf{ACC.WitCreate}(\mathsf{pp}_{\mathsf{ACC}}, \beta, C^{(0)})$, the element β, and the set $C^{(1)}$ to the challenger.

  In the final step, the challenger computes $\mathsf{acc} \leftarrow \mathsf{ACC.Eval}(\mathsf{pp}_{\mathsf{ACC}}, C^{(1)}$, and since $\beta \notin C^{(1)}$ and $\mathsf{ACC.Eval}(\mathsf{pp}_{\mathsf{ACC}}, C^{(0)}) = \mathsf{ACC.Eval}(\mathsf{pp}_{\mathsf{ACC}}, C^{(1)})$, the output of the experiment is 1, which breaks the security of the accumulator.

We have shown in both cases that an adversary which breaks the binding property of the commitment can be used to break either the collision resistance of H or the security of the accumulator, which completes the proof. □

Lemma 6. *The construction* HW *satisfies commitment-conditional soundness, based on the assumption that* H *is collision-resistant and that* ACC *is a secure accumulator.*

Proof. We define the following two intermediate experiments:

- $\mathsf{E}_0(\mathcal{A})$: After sampling $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and computing $(x, t, r, \pi) \leftarrow \mathcal{A}(\mathsf{pp})$, the experiment computes $\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, x; r)$ and outputs $z \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{com}, t, \pi)$.
- $\mathsf{E}_1(\mathcal{A})$: After sampling $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ACC}}, N, b, d) \leftarrow \mathsf{Setup}(1^\lambda)$ and computing $(x, t, r, \pi) \leftarrow \mathcal{A}(\mathsf{pp})$, the experiment computes $\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, x; r)$. Let $C = \{c_1, \ldots, c_d\}$ be the set of values for which $\mathsf{com} = \mathsf{ACC.Eval}(\mathsf{pp}_{\mathsf{ACC}}, C)$. Let $(\mathsf{wit}, \langle v_1, \ldots, v_d \rangle) \leftarrow \pi$, and compute $c^* \leftarrow \mathsf{H}(\mathsf{H}^{t_1}(v_1) \mid\mid \cdots \mid\mid \mathsf{H}^{t_d}(v_d))$. If $c^* \notin C$, then the experiment aborts and immediately outputs 0. Otherwise, the experiment outputs $z \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{com}, t, \pi)$.

Lemma 7. *For all adversaries* $\mathcal{A}$, $\mathsf{E}_0(\mathcal{A}) \approx \mathsf{E}_1(\mathcal{A})$, *based on the security of the accumulator* ACC.

Proof. Note that the only difference between E_0 and E_1 is the conditional check that $c^* \notin C$, resulting the experimenting aborting and outputting 0. Therefore, if $\mathsf{E}_0(\mathcal{A}) = 0$, then $\mathsf{E}_1(\mathcal{A}) = 0$ as well, by definition. If $\mathsf{E}_0(\mathcal{A}) = 1$ and $c^* \in C$, then $\mathsf{E}_1(\mathcal{A}) = 1$ as well. In the following, we consider the only remaining case; let $\mathcal{A}$ be an adversary for which $\mathsf{E}_0(\mathcal{A}) = 1$, $c^* \notin C$, and $\mathsf{E}_1(\mathcal{A}) = 1$.

We construct an algorithm $\mathcal{B}$ which participates in $\mathsf{Expt}^{\mathsf{ACC}}$ as follows. After $\mathcal{B}$ receives $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ACC}}, N, b, d)$ from the challenger, it computes $(x, t, r, \pi) \leftarrow \mathcal{A}(\mathsf{pp})$, with $(\mathsf{wit}, \langle v_1, \ldots, v_d \rangle) \leftarrow \pi$. It then submits a witness wit, the element $c^* \leftarrow \mathsf{H}(\mathsf{H}^{t_1}(v_1) \mid\mid \cdots \mid\mid \mathsf{H}^{t_d}(v_d))$, along with the set C (for which $\mathsf{com} = \mathsf{ACC.Eval}(\mathsf{pp}_{\mathsf{ACC}}, C)$) to the challenger. The challenger responds with a bit z, which is also the output of $\mathcal{B}$.

Now, given that $c^* \notin C$ and yet $\mathsf{E}_1(\mathcal{A}) = 1$, this means that $\mathsf{Expt}^{\mathsf{ACC}}(\mathcal{B}) = 1$ by definition. Hence, using ϵ as the maximum advantage of any adversary participating in $\mathsf{Expt}^{\mathsf{ACC}}$, we have that $|\Pr[\mathsf{E}_0(\mathcal{A}) = 1] - \Pr[\mathsf{E}_1(\mathcal{A}) = 1]| \leq \epsilon$, which yields the claim. □

Lemma 8. *For all adversaries* $\mathcal{A}$, *the quantity* $\Pr[\mathsf{E}_1(\mathcal{A}) = 1] \leq \epsilon$, *where* ϵ *is the probability of finding a collision for* H.

Proof. We construct an algorithm from $\mathsf{E}_1(\mathcal{A})$ which produces a collision for H. Note that if $\mathsf{E}_1(\mathcal{A}) = 1$, then this means that $c^* \in C$, which in turn means that there exists some $w \in \mathsf{MDP}_b(x)$, $r \in \{0,1\}^\lambda$, and $i^* \in [d]$ for which $c^* = \mathsf{HMC}_d(w, \langle \mathsf{H}((i^*, 1) \mid\mid r), \ldots, \mathsf{H}((i^*, d) \mid\mid r))$. Since $w \in \mathsf{MDP}_b(x)$, this implies that $w \leq x < t$. Using $w_1 w_2 \cdots w_d = w$ as the base-b representation of w and $t_1 t_2 \cdots t_d = t$ for t, this means that there exists some index $j \in [d]$ such that $w_j < t_j$.

Now, for $k \in [0, w_j]$, consider the sequence defined as $\alpha_k = \mathsf{H}^{w_j - k}(\mathsf{H}((i^*, j) \mid\mid r))$, and for $k \in [0, t_j]$, the sequence $\beta_k = \mathsf{H}^{t_j - k}(v_j)$. Let k^* be the smallest index for which $\alpha_{k^*} \neq \beta_{k^*}$. There are several cases to consider:

- $k^* = 0$. Then the we output the pair (γ_1, γ_2) as a collision for H, where $\gamma_1 = \mathsf{H}^{w_1}(\mathsf{H}((i^*, 1) \mid\mid r)) \mid\mid \ldots \mid\mid \mathsf{H}^{w_d}(\mathsf{H}((i^*, d) \mid\mid r))$, and $\gamma_2 = \mathsf{H}^{t_1}(v_1) \mid\mid \ldots \mid\mid \mathsf{H}^{t_d}(v_d)$. Note that this is a valid collision, since $\mathsf{H}(\gamma_1) = c^* = \mathsf{H}(\gamma_2)$.
- $0 < k^* \leq w_j$. Then, with $\gamma_1 = \alpha_{k^*}$ and $\gamma_2 = \beta_{k^*}$, we have that (γ_1, γ_2) is a valid collision for H, since $\mathsf{H}(\gamma_1) = \alpha_{k^*-1} = \beta_{k^*-1} = \mathsf{H}(\gamma_2)$.
- Otherwise, with $\gamma_1 = \mathsf{H}((i^*, j) \mid\mid r)$ and $\gamma_2 = \beta_{w_j+1}$, we have that (γ_1, γ_2) is a valid collision for H, since $w_j < t_j$ implies that β_{w_j+1} is well-defined, and $\mathsf{H}(\beta_{w_j+1}) = \beta_{w_j} = \alpha_{w_j} = \mathsf{H}(\gamma_1)$ by definition.

We have shown that in all cases, we are able to produce a collision for H, which yields the claim. □

Note that E_0 corresponds exactly to the experiment defined in Definition 3 when instantiated with HW. Thus, putting together Lemmas 7 and 8 yields the claim. □

Lemma 9. *The construction* HW *satisfies witness indistinguishability, based on the assumption that* H *is random-output secure.*

Proof. We describe a simulator $\mathsf{Sim} : \mathsf{PP} \times [N] \to \mathbb{P} \times \mathbb{C}$ which takes as input the public parameters $\mathsf{pp} \in \mathsf{PP}$ and an integer threshold $t \in [N]$, and outputs a tuple $(\pi, \mathsf{com}) \in \mathbb{P} \times \mathbb{C}$ as follows:

1. Using $(\mathsf{pp}_{\mathsf{ACC}}, N, b, d) \leftarrow \mathsf{pp}$, Sim samples a random index $i^* \leftarrow_{\mathsf{R}} [d]$, $d - 1$ values $c_1, \ldots, c_{i^*-1}, c_{i^*+1}, \ldots, c_d \leftarrow_{\mathsf{R}} \{0,1\}^\lambda$, and values $v_1, \ldots, v_d \leftarrow_{\mathsf{R}} \{0,1\}^\lambda$.
2. With $t_1 \cdots t_d$ as the base-b representation of t, for each $j \in [d]$, define $c_{i^*} \leftarrow \mathsf{H}(\mathsf{H}^{t_1}(v_1) \mid\mid \cdots \mid\mid \mathsf{H}^{t_d}(v_d))$.
3. Using $C = \{c_1, \ldots, c_d\}$, the simulator computes $\mathsf{wit} \leftarrow (\mathsf{ACC.WitCreate}(\mathsf{pp}_{\mathsf{ACC}}, c_{i^*}, C)$, $\pi \leftarrow (\mathsf{wit}, \langle v_1, \ldots, v_d \rangle)$, $\mathsf{com} \leftarrow \mathsf{ACC.Eval}(\mathsf{pp}_{\mathsf{ACC}}, C)$, and outputs (π, com).

In order to prove witness indistinguishability, for fixed integers $x, t \in [N]$ with $x \geq t$, we define the following series of distributions:

- D_0: This is equal to the distribution $(\mathsf{pp}, \mathsf{Sim}(t), x, t)$.
- D_1: This is the same as D_0, except in how the values $v_1, \ldots, v_d$ are generated. Instead of uniformly sampling them from $\{0,1\}^\lambda$, we instead sample $r^* \leftarrow_\mathsf{R} \{0,1\}^\lambda$, and do the following: Let $\langle w_1, \ldots, w_m \rangle \leftarrow \mathsf{MDP}_b(x)$, and let $i^* \in [m]$ be the first index for which w_{i^*} dominates t. Denoting $y_1 \cdots y_d$ as the base-b representation of w_{i^*} and $t_1 \cdots t_d$ as the base-b representation of t, for each $j \in [d]$, define $v_j \leftarrow \mathsf{H}^{y_j - t_j}(\mathsf{H}((i^*, j) \mid\mid r^*))$. All other steps remain the same.
- D_2: This is the same as D_1, except in how the values $c_1, \ldots, c_{i^*-1}, c_{i^*+1}, \ldots, c_d$ are generated. Instead of uniformly sampling them from $\{0,1\}^\lambda$, we sample $r \leftarrow_\mathsf{R} \{0,1\}^\lambda$, for each $i, j \in [d]$ compute $r_{i,j} \leftarrow \mathsf{H}((i,j) \mid\mid r)$, compute $\langle w_1, \ldots, w_m \rangle \leftarrow \mathsf{MDP}_b(x)$, use H and r to select a permutation $P' : [d] \setminus \{i^*\} \to [d] \setminus \{i^*\}$, and for each $i \in [d] \setminus \{i^*\}$, set $a_i \leftarrow \mathsf{HWS}_d(w_i; r_i)$ and finally $c_i \leftarrow a_{P'(i)}$. All other steps remain the same.
- D_3: This is equal to the distribution $(\mathsf{pp}, (\mathsf{Prove}(\mathsf{pp}, x, t; r), \mathsf{Commit}(\mathsf{pp}, x; r)), x, t)$.

Lemma 10. *For every adversary that can distinguish* D_0 *from* D_1 *with advantage at most* ϵ*, there exists an adversary that can break the random-output security of* H *with advantage at least* $\epsilon/((b+1)d)$.

Proof. Let Q_1 represent the total number of evaluations of H needed to produce the values $v_1, \ldots, v_d$ in D_1. We define a series of hybrid distributions $F_0, \ldots, F_{Q_1}$ such that for each $i \in [0, Q_1]$, F_i is the same as D_1, except that a random output $\{0,1\}^\lambda$ is used in place of the first i evaluations of H in D_1. Note that the evaluations of H can be ordered so that F_i is always well-defined. Also, observe that F_0 is identical to D_1, and F_M is identical to D_0.

For each $i \in [Q_1]$, any adversary which can distinguish F_{i-1} from F_i with advantage ϵ can then be used to break random-output security for the hash function with advantage ϵ. We then sum the advantages across each F_i, along with using the fact that $Q_1 \leq (b+1)d$, to conclude the proof. □

Lemma 11. *For every adversary that can distinguish* D_1 *from* D_2 *with advantage at most* ϵ*, there exists an adversary that can break the random-output security of* H *with advantage at least* $\epsilon/(bd(d-1))$.

Proof. This proof follows almost identically from the format of the proof for Lemma 10, but we repeat it here for completeness. Let Q_2 represent the total number of evaluations of H needed to produce the values $c_1, \ldots, c_{i^*-1}, c_{i^*+1}, \ldots, c_d$. We define a series of hybrid distributions $F_0, \ldots, F_M$ such that for each $i \in [0, Q_2]$, F_i is the same as D_2, except that a random output $\{0,1\}^\lambda$ is used in place of the first i evaluations of H in D_2. Note that the evaluations of H can be ordered so that F_i is always well-defined. Also, observe that F_0 is identical to D_2, and F_M is identical to D_1.

For each $i \in [Q_2]$, any adversary which can distinguish F_{i-1} from F_i with advantage ϵ can then be used to break random-output security for the hash function with advantage ϵ. We then sum the advantages across each F_i, along with using the fact that $Q_2 \leq (b+1)d^2$, to conclude the proof. □

Lemma 12. *The two distributions* D_2 *and* D_3 *are identical.*

Proof. The only distinction between the two distributions is that in D_2, a random index $i^* \in [d]$ is selected, and used to order a set of d values along with a permutation $P' : [d] \setminus \{i^*\} \to [d] \setminus \{i^*\}$, whereas in D_3, a permutation $P : [d] \to [d]$ is used. Since the permutation induced by P' and the random selection of the index i^* is identical to having selected P from the entire domain $[d]$, we conclude that D_2 and D_3 are indeed identical. □

By putting together Lemmas 10, 11, and 12, we can conclude that an adversary for $\mathsf{Expt}^{\mathsf{WI}}$ that can interact with a challenger that simply computes $(\pi, \mathsf{com}) \leftarrow \mathsf{Sim}(\mathsf{pp}, t)$ on input (x, t) in Step 2 of $\mathsf{Expt}^{\mathsf{WI}}$, and this response is indistinguishable from a "real world" computation of π and com using Prove and Commit, whilst being independent of x. Therefore, if δ represents the maximum advantage of any adversary in breaking random-output security for H, then any adversary can have advantage at most $\delta + bd + 2d^2$ in breaking witness indistinguishability for the construction HW. □

Finally, with Lemmas 4, 5, 6, and 9, we conclude the proof of Theorem 2.

B Full HashWires Snapshot

We present a snapshot of the optimized HashWires commitment for the integer 312 (base-4), using deterministic KDFs and shuffling, MDP wiring, and both PLA and padded sparse Merkle tree accumulators.

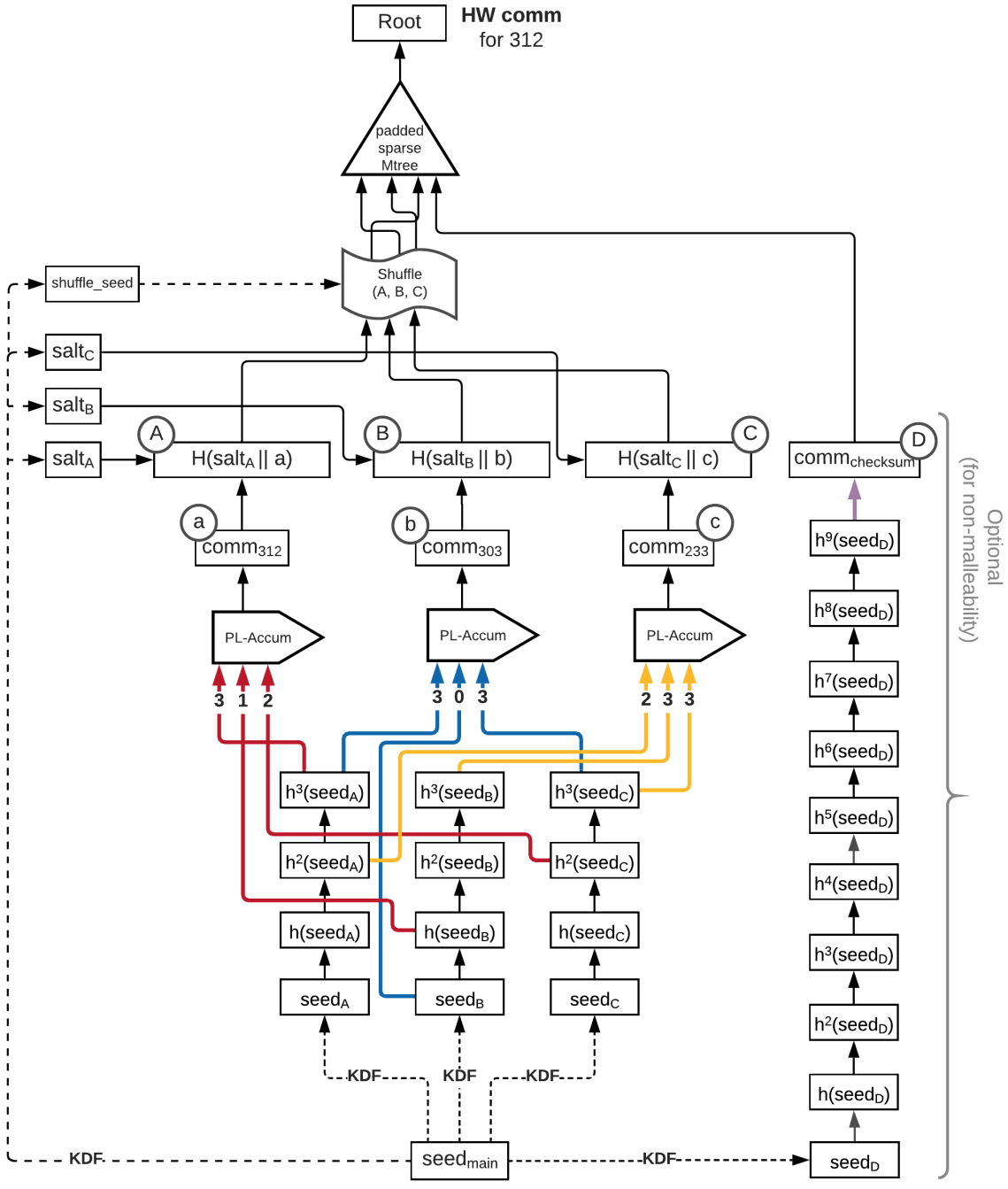

Fig. 6. Full snapshot of a HashWires commitment structure for 312 in base-4, including a single chain for malleability checksum. In our benchmarks, we used this HashWires design using salted hash functions as KDFs, Durstenfeld's algorithm for shuffling and we did not include the optional checksum chain.