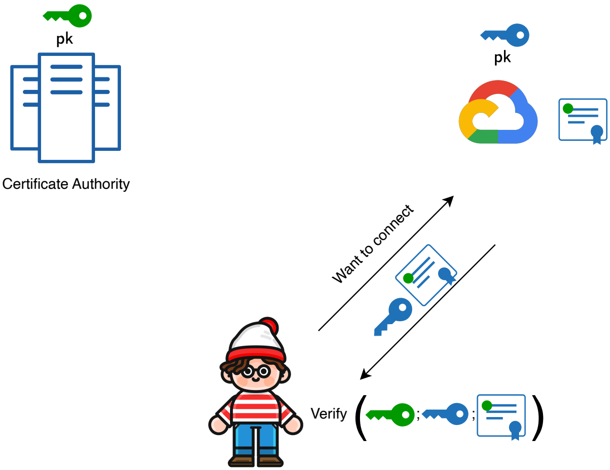
RSA-Based Dynamic Accumulator without Hashing into Primes

Victor Youdom Kemmoe   Anna Lysyanskaya

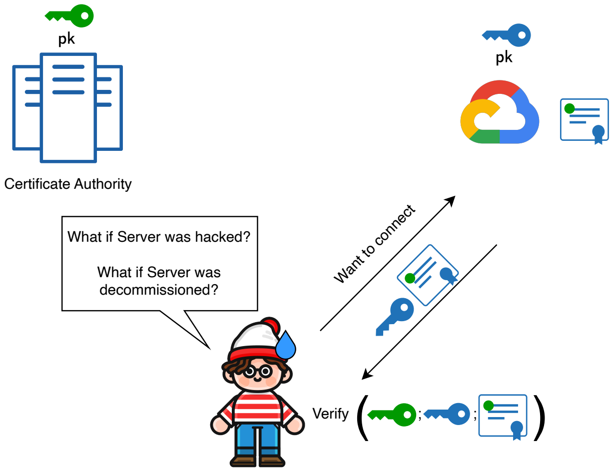
Brown University

ACM CCS 2024

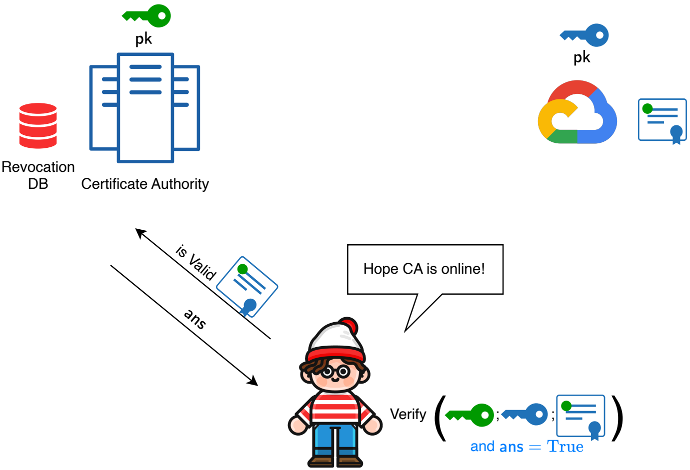
Motivation (Case of WebPKI certificate revocation)



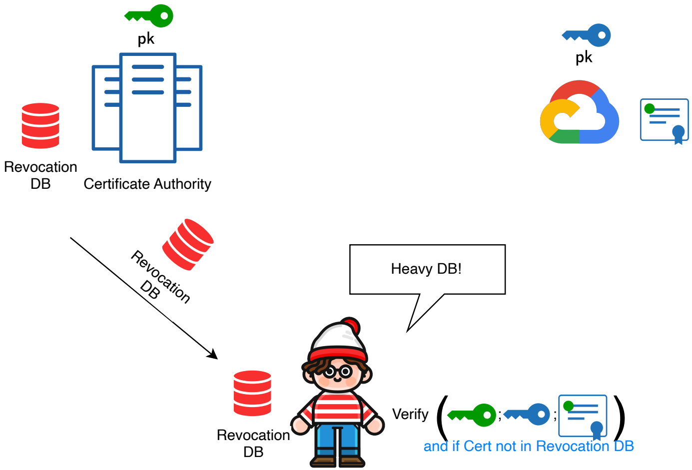
Motivation (Case of WebPKI certificate revocation)



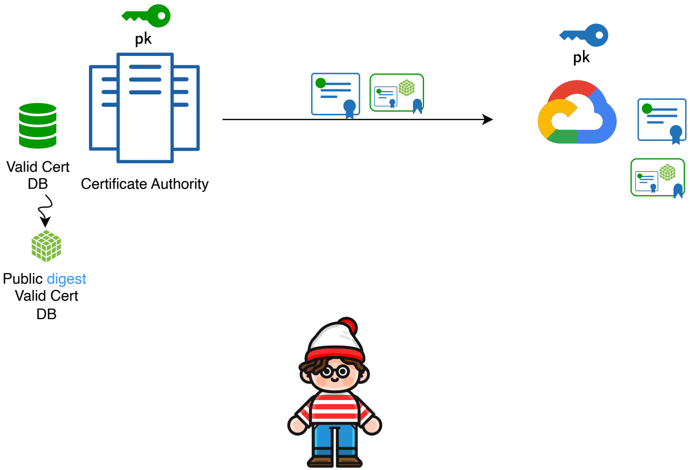
Motivation (Case of WebPKI certificate revocation)



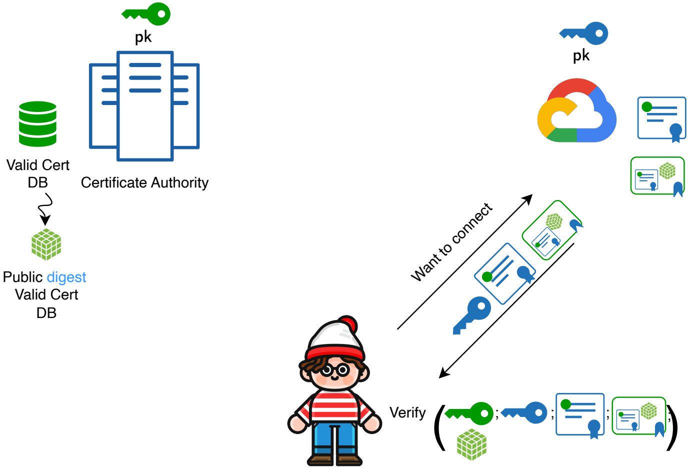
Motivation (Case of WebPKI certificate revocation)



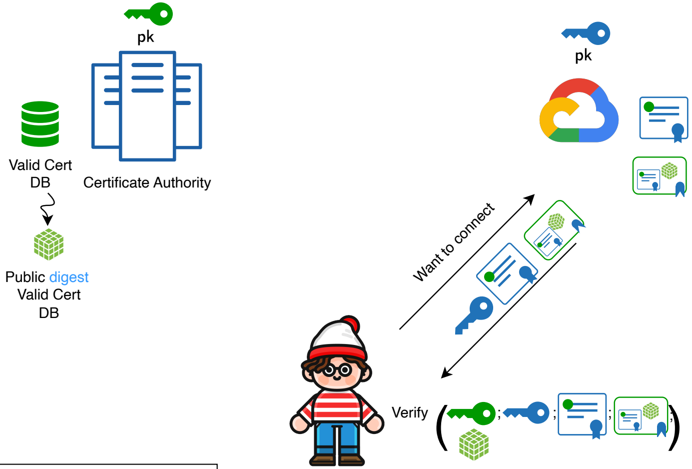
Motivation (Case of WebPKI certificate revocation)



Motivation (Case of WebPKI certificate revocation)



Motivation (Case of WebPKI certificate revocation)

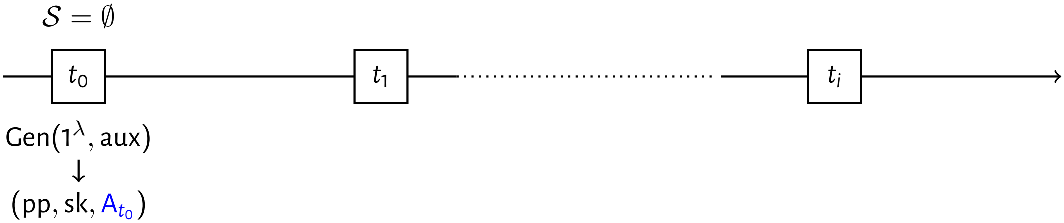


CA is not required to be always online

Digest has a small (constant) size

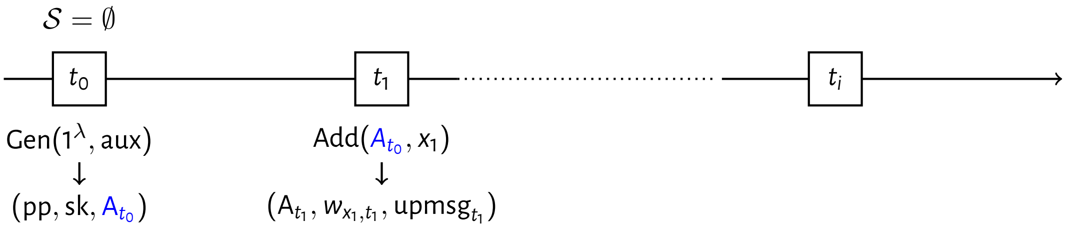
Dynamic Accumulator

Syntax [DHS15, BCD+17, BKR23]



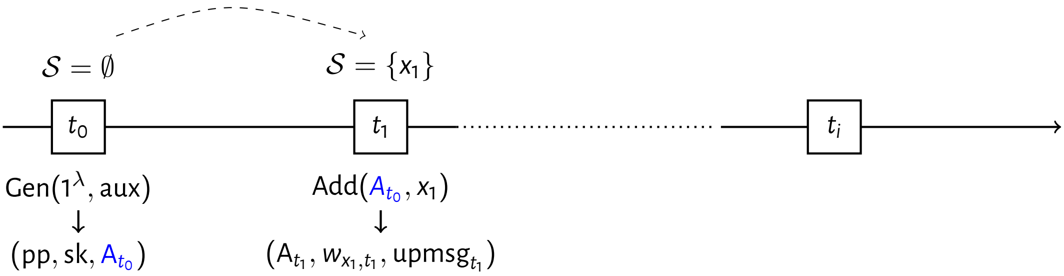
Dynamic Accumulator

Syntax [DHS15, BCD+17, BKR23]



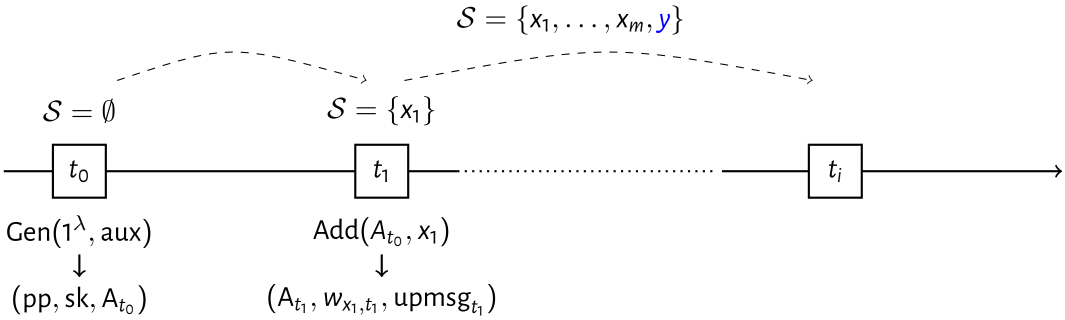
Dynamic Accumulator

Syntax [DHS15, BCD+17, BKR23]



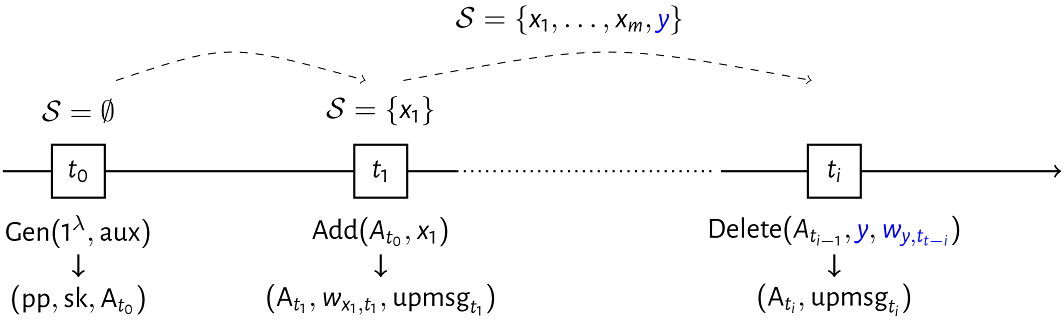
Dynamic Accumulator

Syntax [DHS15, BCD+17, BKR23]



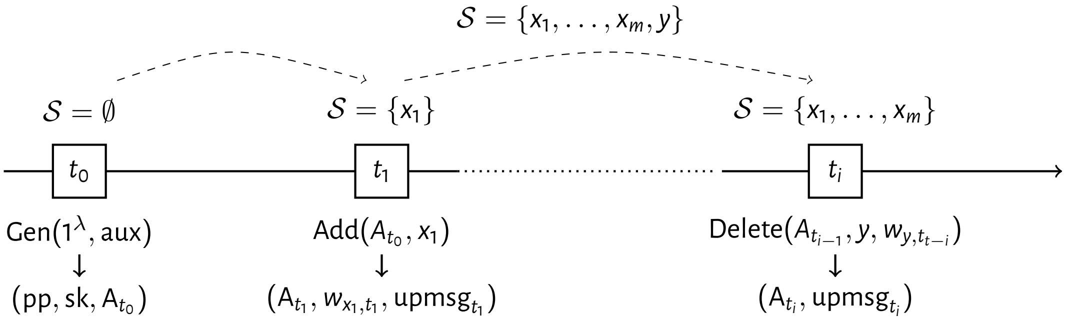
Dynamic Accumulator

Syntax [DHS15, BCD+17, BKR23]



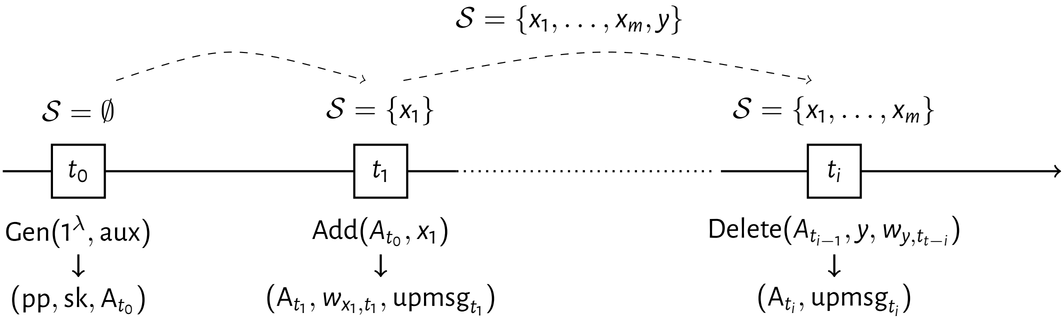
Dynamic Accumulator

Syntax [DHS15, BCD+17, BKR23]



Dynamic Accumulator

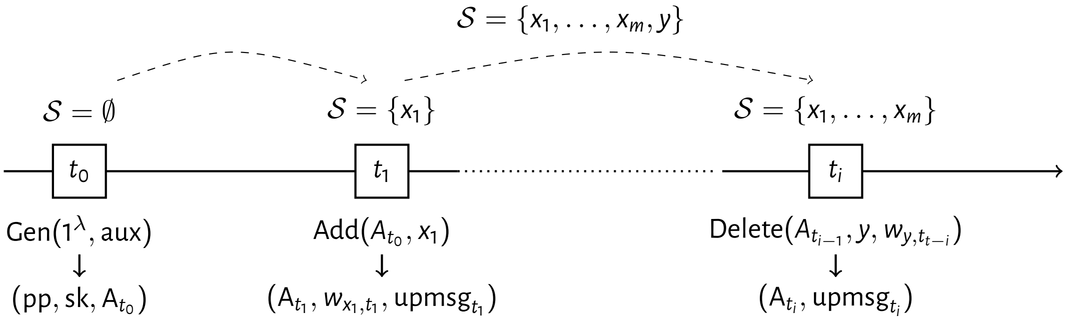
Syntax [DHS15, BCD+17, BKR23]



- $\mathsf{MemWitUp}(x, w_{x,t}, \mathsf{upmsg}_{t+1}) \rightarrow w_{x,t+1}$
- $\mathsf{MemVerify}(\mathsf{A}_t, x, w_{x,t}) \rightarrow \mathsf{Accept}/\mathsf{Reject}$

Dynamic Accumulator

Syntax [DHS15, BCD+17, BKR23]



- $\mathsf{MemWitUp}(x, w_{x,t}, \mathsf{upmsg}_{t+1}) \to w_{x,t+1}$
- $\mathsf{MemVerify}(\mathsf{A}_t, x, w_{x,t}) \to \mathsf{Accept}/\mathsf{Reject}$
- $\mathsf{NonMemWitCreate}(\mathsf{A}_t, x, \{\mathsf{upmsg}_i\}_{i=1}^{t}) \to \bar{w}_{x,t}$
- $\mathsf{NonMemWitUp}(x, \bar{w}_{x,t}, \mathsf{upmsg}_{t+1}) \to \bar{w}_{x,t+1}$
- $\mathsf{NonMemVerify}(\mathsf{A}_t, x, \bar{w}_{x,t}) \to \mathsf{Accept}/\mathsf{Reject}$

Dynamic Accumulator's Properties

Definition (Compactness)

$|\mathsf{A}| = \mathsf{poly}(\lambda), |w_{x,t}| = |\bar{w}_{x,t}| = \mathsf{poly}(\lambda, |x|)$

Dynamic Accumulator's Properties

Definition (Compactness)

$|\mathsf{A}| = \mathrm{poly}(\lambda), |w_{x,t}| = |\bar{w}_{x,t}| = \mathrm{poly}(\lambda, |x|)$

Definition (Correctness–Informal)

An accumulator scheme is correct if given A_t, $(x, w_{x,t})$, and $(y, \bar{w}_{y,t})$ such that $w_{x,t}, \bar{w}_{y,t}$ are up-to-date:

- $(x, w_{x,t})$ pass MemVerify with overwhelming probability
- $(y, \bar{w}_{y,t})$ pass NonMemVerify with overwhelming probability

Dynamic Accumulator's Properties

Definition (Compactness)

$|\mathsf{A}| = \mathrm{poly}(\lambda)$, $|w_{x,t}| = |\bar{w}_{x,t}| = \mathrm{poly}(\lambda, |x|)$

Definition (Correctness–Informal)

An accumulator scheme is correct if given A_t, $(x, w_{x,t})$, and $(y, \bar{w}_{y,t})$ such that $w_{x,t}, \bar{w}_{y,t}$ are up-to-date:

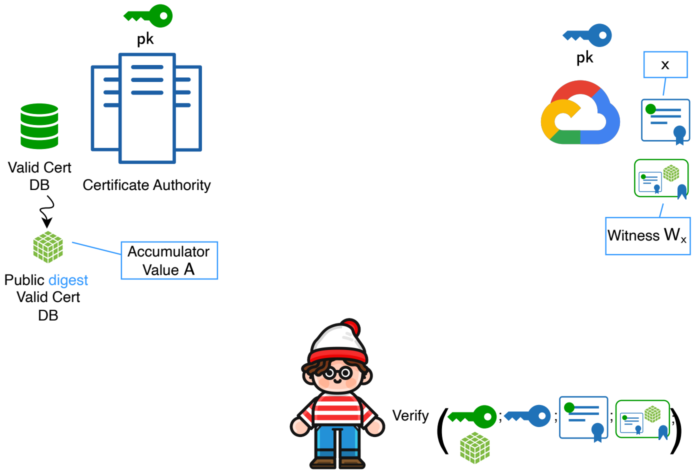
- $(x, w_{x,t})$ pass MemVerify with overwhelming probability
- $(y, \bar{w}_{y,t})$ pass NonMemVerify with overwhelming probability

Definition (Security–Informal)

An accumulator scheme is secure if for all poly-time adversary $\mathcal{A}$:

- It is *hard* to output a valid w_x for any $x \notin \mathcal{S}$
- It is *hard* to output a valid $\bar{w}_y$ for any $y \in \mathcal{S}$

Dynamic Accumulator in WebPKI Certificate Revocation



Prior Works

- Benaloh and de Mare [BdM94]: a static positive accumulator for random integers based on the hardness of computing arbitrary roots in RSA groups.

$$A \leftarrow u^{\prod_{i=1}^{n} x_i} \bmod n \quad \text{and} \quad w_{x_i} = A^{1/x_i}$$

Prior Works

- Benaloh and de Mare [BdM94]: a static positive accumulator for random integers based on the hardness of computing arbitrary roots in RSA groups.

$$A \leftarrow u^{\prod_{i=1}^{n} x_i} \bmod n \quad \text{and} \quad w_{x_i} = A^{1/x_i}$$

- Barić and Pfitzmann [BP97]: improved upon [BdM94] by changing the domain of accumulated elements to PRIMES and proving the security of their proposal under the *strong RSA assumption*.

Prior Works

- Benaloh and de Mare [BdM94]: a static positive accumulator for random integers based on the hardness of computing arbitrary roots in RSA groups.

$$A \leftarrow u^{\prod_{i=1}^{n} x_i} \bmod n \quad \text{and} \quad w_{x_i} = A^{1/x_i}$$

- Barić and Pfitzmann [BP97]: improved upon [BdM94] by changing the domain of accumulated elements to PRIMES and proving the security of their proposal under the *strong RSA assumption*.
- Camenisch and Lysyanskaya [CL02] showed how to obtain a positive dynamic accumulator from [BP97], and Li, Li and Xue [LLX07] showed how to obtain a universal dynamic accumulator from [CL02].

Prior Works

- Benaloh and de Mare [BdM94]: a static positive accumulator for random integers based on the hardness of computing arbitrary roots in RSA groups.

$$A \leftarrow u^{\prod_{i=1}^{n} x_i} \bmod n \quad \text{and} \quad w_{x_i} = A^{1/x_i}$$

- Barić and Pfitzmann [BP97]: improved upon [BdM94] by changing the domain of accumulated elements to PRIMES and proving the security of their proposal under the *strong RSA assumption*.
- Camenisch and Lysyanskaya [CL02] showed how to obtain a positive dynamic accumulator from [BP97], and Li, Li and Xue [LLX07] showed how to obtain a universal dynamic accumulator from [CL02].

Working over PRIMES requires hashing to prime integers in practice:

Try $r \in \{0, \ldots, N\}$ until $H(x; r)$ is prime. Then, accumulate $A' \leftarrow A^{H(x;r)}$.

Based on the Prime Number Theorem, this incurs an overhead of $O(\log N)$

Prior Works

- Other proposals based on billinear-pairing [Ngu05, CKS09, ATSM09] that works over integers.
  - Require public parameters whose size is linear in the number of elements to be accumulated.
  - Reducing the size of public parameters is possible at the expense of requiring a trapdoor for Add and Delete.
- Other proposals based on Merkle-tree [CW09, RY16] that works over integers.
  - Witness size is logarithmic in the number of elements accumulated.
  - Supporting non-membership and deletion is non-trivial.

Our Contributions

- RSA-based unviersal and positive dynamic accumulators defined over large odd integers.
- Security holds under the strong RSA assumption in the random oracle model.
- **At least** three times faster than RSA-based accumulators defined over primes for $\lambda = 128$.
- A variant of Wesolowski's Proof of Exponentation [Wes20], called *SimPoE*, that does not require hashing to primes.
- We showed how to aggregate (non-)membership witnesses and use *SimPoE* to reduce the verification time of aggregated witnesses.

Outline

1. Universal Accumulator: Gen, Add, Delete, MemVerify, MemWitUp
2. Wesolowski's Proof of Exponentation without hashing to primes, i.e., SimPoE

Some Number Theory notions

- $\mathsf{Odds}(2^{\ell-1}, 2^{\ell}-1) \stackrel{\text{def}}{=} \{2^{\ell-1} \leq n \leq 2^{\ell}-1 : n \bmod 2 = 1\}$.
- $P^{+}(a)$: return the largest prime factor of a.

Based on [dB51, HT93],

Lemma (Informal)

Given a sufficiently large $\ell \in \mathbb{N}$, *If* $a \leftarrow\$ \mathsf{Odds}(2^{\ell-1}, 2^{\ell}-1)$, *Then*

$$\Pr\left[P^{+}(a) > 2^{\sqrt[4]{\ell}}\right] \geq 1 - O\left(2^{-\sqrt[4]{\ell}}\right)$$

Some Number Theory notions

- $\text{Odds}(2^{\ell-1}, 2^{\ell}-1) \stackrel{\text{def}}{=} \{2^{\ell-1} \leq n \leq 2^{\ell}-1 : n \bmod 2 = 1\}$.
- $P^{+}(a)$: return the largest prime factor of a.

Based on [dB51, HT93],

Lemma (Informal)

Given a sufficiently large $\ell \in \mathbb{N}$*, If* $a \leftarrow_{\$} \text{Odds}(2^{\ell-1}, 2^{\ell}-1)$*, Then*

$$\Pr\left[P^{+}(a) > 2^{\sqrt[4]{\ell}}\right] \geq 1 - O\left(2^{-\sqrt[4]{\ell}}\right)$$

In other words, If $a \leftarrow_{\$} \text{Odds}(2^{\ell-1}, 2^{\ell}-1)$, then with overwhelming probability, a has a large prime factor with $\Omega(\sqrt[4]{\ell})$-bits

Some Number Theory notions

Corollary (Informal)

For $m \in \mathbb{N}$, *given a sufficiently large* $\ell \in \mathbb{N}$, *and* $a_1, a_2, \ldots, a_m \sim U\left(\mathrm{Odds}(2^{\ell-1}, 2^{\ell}-1)\right)$. *Then,*

$$\Pr\left[P^{+}(a_i) \mid \prod_{j \in [m] \setminus \{i\}} a_j\right] \leq m^2 O\left(2^{-\sqrt[4]{\ell}}\right) \quad \forall i \in [m]$$

Some Number Theory notions

Corollary (Informal)

For $m \in \mathbb{N}$, given a sufficiently large $\ell \in \mathbb{N}$, and $a_1, a_2, \ldots, a_m \sim U\left(\text{Odds}(2^{\ell-1}, 2^{\ell}-1)\right)$. Then,

$$\Pr\left[P^{+}(a_i) \mid \prod_{j \in [m] \setminus \{i\}} a_j\right] \leq m^2 O\left(2^{-\sqrt[4]{\ell}}\right) \quad \forall i \in [m]$$

In other words, If you select $a_1, \ldots, a_m \in \text{Odds}(2^{\ell-1}, 2^{\ell}-1)$ uniformly at random, then the probability that $a_i \mid \prod_{j=1, j \neq i}^{m} a_j$ is negligible

Our Universal Accumulator Construction

$$H : \{0,1\}^* \to \mathrm{Odds}(2^{\ell-1}, 2^{\ell} - 1)$$

$\ell = \mathrm{poly}(\lambda)$, s.t. for all $x \in \{0,1\}^*$, $\mathbf{P}^{+}(\mathbf{H}(\mathbf{x})) > \mathbf{2}^{\sqrt[4]{\ell}}$ with overwhelming probability.

Our Universal Accumulator Construction

$$H : \{0,1\}^* \to \mathsf{Odds}(2^{\ell-1}, 2^{\ell} - 1)$$

$\ell = \mathsf{poly}(\lambda)$, s.t. for all $x \in \{0,1\}^*$, $\mathbf{P}^+(\mathbf{H}(\mathbf{x})) > \mathbf{2}^{\sqrt[4]{\ell}}$ with overwhelming probability.

- $\mathsf{Gen}(1^\lambda, \perp)$:
  1. Return $\mathsf{pp} = (n, u)$, $\mathsf{sk} = (p-1)(q-1)$, and $\mathsf{A}_0 \leftarrow u \in \mathrm{QR}_n$.
- $\mathsf{Add}(\mathsf{pp}, \mathsf{A}, x)$:
  1. Parse pp as (n, u).
  2. Compute $\mathsf{A}' \leftarrow \mathsf{A}^{\mathbf{H}(\mathbf{x})} \bmod n$.
  3. Let $\mathbf{s} = (1)$, $w_x = (\mathsf{A}, \mathbf{s})$ and $\mathsf{upmsg} = (\mathsf{add}, H(x), \mathbf{1}, \mathsf{A}, \mathsf{A}')$.
  4. Return A', w_x, and upmsg.

Our Universal Accumulator Construction

- $\mathsf{Delete}(\mathsf{pp}, \mathsf{sk}, \mathsf{A}, x, w_x)$:
  1. Parse pp as (n, u)
  2. Compute $\gamma \leftarrow 1/H(x) \bmod \mathsf{sk}$, and let $\delta = 1$.
  3. Compute $\mathsf{A}' \leftarrow \mathsf{A}^{\gamma} \bmod n$.
  4. Let upmsg $= (\mathsf{del}, H(x), \delta, \mathsf{A}, \mathsf{A}')$.
  5. Return A', and upmsg.

Our Universal Accumulator Construction

- Delete(pp, sk, A, x, w_x):
  1. Parse pp as (n, u)
  2. Compute $\gamma \leftarrow 1/H(x) \bmod \mathsf{sk}$, and let $\delta = 1$.
  3. Compute $\mathsf{A}' \leftarrow \mathsf{A}^{\gamma} \bmod n$.
  4. Let upmsg $= (\mathsf{del}, H(x), \delta, \mathsf{A}, \mathsf{A}')$.
  5. Return A′, and upmsg.

Note: We can use w_x to avoid using sk

How to update membership witnesses

We recall Camenisch-Lysyanskaya [CL02] membership update algorithm.

- $\mathsf{MemWitUp}(\mathsf{pp}, x, w_x, \mathsf{upmsg})$:
  1. Parse pp as (n, u), w_x as $(\mathtt{w}, \mathbf{s})$, and upmsg as $(\mathsf{op}, H(y), \delta, \mathsf{A}, \mathsf{A}')$.
  2. If $\mathsf{op} = \mathsf{add}$, *// After* Add, $\mathsf{A}' = \mathsf{A}^{H(y)}$
     - compute $\mathtt{w}' \leftarrow \mathtt{w}^{H(y)} \bmod n$, and let $w'_x = (\mathtt{w}', \mathbf{s})$.

How to update membership witnesses

We recall Camenisch-Lysyanskaya [CL02] membership update algorithm.

- $\mathsf{MemWitUp}(\mathsf{pp}, x, w_x, \mathsf{upmsg})$:
  1. Parse pp as (n, u), w_x as $(\mathtt{w}, \mathbf{s})$, and upmsg as $(\mathsf{op}, H(y), \delta, \mathsf{A}, \mathsf{A}')$.
  2. If $\mathsf{op} = \mathsf{add}$,
     - compute $\mathtt{w}' \leftarrow \mathtt{w}^{H(y)} \bmod n$, and let $w_x' = (\mathtt{w}', \mathbf{s})$.
  3. Else if $\mathsf{op} = \mathsf{del}$, do: // *After* Delete, $\mathsf{A}' = \mathsf{A}^{1/H(y)}$
     1. Compute $a, b \in \mathbb{Z}$ such that $aH(x) + bH(y) = \gcd(H(x), H(y))$.
     2. Compute $\mathtt{w}' \leftarrow (\mathsf{A}')^a \mathtt{w}^b \bmod n$.

How to update membership witnesses

We recall Camenisch-Lysyanskaya [CL02] membership update algorithm.

- $\mathsf{MemWitUp}(\mathsf{pp}, x, w_x, \mathsf{upmsg})$:
  1. Parse pp as (n, u), w_x as $(\mathtt{w}, \mathbf{s})$, and upmsg as $(\mathsf{op}, H(y), \delta, \mathsf{A}, \mathsf{A}')$.
  2. If $\mathsf{op} = \mathsf{add}$,
     - compute $\mathtt{w}' \leftarrow \mathtt{w}^{H(y)} \bmod n$, and let $w'_x = (\mathtt{w}', \mathbf{s})$.
  3. Else if $\mathsf{op} = \mathsf{del}$, do: // *After* Delete, $\mathsf{A}' = \mathsf{A}^{1/H(y)}$
     1. Compute $a, b \in \mathbb{Z}$ such that $aH(x) + bH(y) = \gcd(H(x), H(y))$.
     2. Compute $\mathtt{w}' \leftarrow (\mathsf{A}')^a \mathtt{w}^b \bmod n$.

Correctness issue

$$\underset{\sim\sim}{(\mathtt{w}')^{H(x)}} = ((\mathsf{A}')^a \mathtt{w}^b)^{H(x)} = ((\mathsf{A}')^a \mathtt{w}^b)^{H(x)H(y)(1/H(y))} = (\mathsf{A}^{1/H(y)})^{\gcd(H(x),H(y))} = \underset{\sim\sim}{(\mathsf{A}')^{\gcd(H(x),H(y))}}$$

How to update membership witnesses

We recall Camenisch-Lysyanskaya [CL02] membership update algorithm.

- MemWitUp(pp, x, w_x, upmsg):
  1. Parse pp as (n, u), w_x as $(\mathtt{w}, \mathbf{s})$, and upmsg as $(\mathsf{op}, H(y), \delta, \mathsf{A}, \mathsf{A}')$.
  2. If op $=$ add,
     - compute $\mathtt{w}' \leftarrow \mathtt{w}^{H(y)} \bmod n$, and let $w'_x = (\mathtt{w}', \mathbf{s})$.
  3. Else if op $=$ del, do: *// After* Delete, $\mathsf{A}' = \mathsf{A}^{1/H(y)}$
     1. Compute $a, b \in \mathbb{Z}$ such that $aH(x) + bH(y) = \gcd(H(x), H(y))$.
     2. Compute $\mathtt{w}' \leftarrow (\mathsf{A}')^a \mathtt{w}^b \bmod n$.

Correctness issue

$$(\mathtt{w}')^{H(x)} = ((\mathsf{A}')^a \mathtt{w}^b)^{H(x)} = ((\mathsf{A}')^a \mathtt{w}^b)^{H(x)H(y)(1/H(y))} = (\mathsf{A}^{1/H(y)})^{\gcd(H(x),H(y))} = (\mathsf{A}')^{\gcd(H(x),H(y))}$$

Fixing the issue

Observe that $(\mathtt{w}')^{\mathbf{H(x)}/\gcd(\mathbf{H(x)},\mathbf{H(y)})} = \mathsf{A}'$. What if we *consider* $\frac{H(x)}{\gcd(H(x),H(y))}$ as our accumulated element?

How to update membership witnesses

We recall Camenisch-Lysyanskaya [CL02] membership update algorithm.

- $\mathsf{MemWitUp}(\mathsf{pp}, x, w_x, \mathsf{upmsg})$:
  1. Parse pp as (n, u), w_x as $(\mathtt{w}, \mathbf{s})$, and upmsg as $(\mathsf{op}, H(y), \delta, \mathsf{A}, \mathsf{A}')$.
  2. If $\mathsf{op} = \mathsf{add}$,
     - compute $\mathtt{w}' \leftarrow \mathtt{w}^{H(y)} \bmod n$, and let $w'_x = (\mathtt{w}', \mathbf{s})$.
  3. Else if $\mathsf{op} = \mathsf{del}$, do: // *After* Delete, $\mathsf{A}' = \mathsf{A}^{1/H(y)}$
     1. Compute $a, b \in \mathbb{Z}$ such that $aH(x) + bH(y) = \gcd(H(x), H(y))$.
     2. Compute $\mathtt{w}' \leftarrow (\mathsf{A}')^a \mathtt{w}^b \bmod n$.

Correctness issue

$$(\mathtt{w}')^{H(x)} = ((\mathsf{A}')^a \mathtt{w}^b)^{H(x)} = ((\mathsf{A}')^a \mathtt{w}^b)^{H(x)H(y)(1/H(y))} = (\mathsf{A}^{1/H(y)})^{\gcd(H(x),H(y))} = (\mathsf{A}')^{\gcd(H(x),H(y))}$$

Fixing the issue

Observe that $(\mathtt{w}')^{\mathbf{H(x)}/\gcd(\mathbf{H(x)},\mathbf{H(y)})} = \mathsf{A}'$. What if we *consider* $\frac{H(x)}{\gcd(H(x),H(y))}$ as our accumulated element?

Note that $P^+(H(x)) = P^+(\frac{H(x)}{\gcd(H(x),H(y))})$.

Our Universal Accumulator Construction

We have x with witness $w_x = (\mathtt{w}, \mathbf{s})$. *Assume* $\mathbf{s} = ()$

- y_1 was deleted
  Compute $a_1, b_1 \in \mathbb{Z}$ such that $a_1 H(x) + b_1 H(y_1) = \gcd(H(x), H(y_1))$
  Compute $\mathtt{w}' \leftarrow (\mathrm{A}')^{a_1} \mathtt{w}^{b_1} \bmod n$, and $\mathbf{s}' \leftarrow \mathbf{s} \| (\gcd(H(x), H(y_1)))$

Our Universal Accumulator Construction

We have x with witness $w_x = (\mathtt{w}, \mathbf{s})$. *Assume* $\mathbf{s} = ()$

- y_1 was deleted
  Compute $a_1, b_1 \in \mathbb{Z}$ such that $a_1 H(x) + b_1 H(y_1) = \gcd(H(x), H(y_1))$
  Compute $\mathtt{w}' \leftarrow (\mathsf{A}')^{a_1} \mathtt{w}^{b_1} \bmod n$, and $\mathbf{s}' \leftarrow \mathbf{s} \| (\gcd(H(x), H(y_1)))$

  To verify, parse $\mathbf{s}'$ as $\gcd(H(x), H(y_1))$, compute $\mathtt{x} \leftarrow \frac{H(x)}{\gcd(H(x),H(y_1))}$. Finally, check

$$(\mathtt{w}')^{\mathtt{x}} \stackrel{?}{=} \mathsf{A}'$$

Our Universal Accumulator Construction

We have x with witness $w_x = (\mathtt{w}, \mathbf{s})$. *Assume* $\mathbf{s} = ()$

- y_1 was deleted
  Compute $a_1, b_1 \in \mathbb{Z}$ such that $a_1 H(x) + b_1 H(y_1) = \gcd(H(x), H(y_1))$
  Compute $\mathtt{w}' \leftarrow (\mathsf{A}')^{a_1} \mathtt{w}^{b_1} \mod n$, and $\mathbf{s}' \leftarrow \mathbf{s} \| (\gcd(H(x), H(y_1)))$

  To verify, parse $\mathbf{s}'$ as $\gcd(H(x), H(y_1))$, compute $\mathtt{x} \leftarrow \frac{H(x)}{\gcd(H(x), H(y_1))}$. Finally, check

$$(\mathtt{w}')^{\mathtt{x}} \stackrel{?}{=} \mathsf{A}'$$

- y_2 was deleted
  Compute $a_2, b_2 \in \mathbb{Z}$ such that $a_2 \mathtt{x} + b_2 H(y_1) = \gcd(\mathtt{x}, H(y_2))$
  Compute $\mathtt{w}'' \leftarrow (\mathsf{A}'')^{a_2} \mathtt{w}^{b_2} \mod n$, and $\mathbf{s}'' \leftarrow \mathbf{s}' \| (\gcd(\mathtt{x}, H(y_2)))$

Our Universal Accumulator Construction

We have x with witness $w_x = (\mathtt{w}, \mathbf{s})$. *Assume* $\mathbf{s} = ()$

- y_1 was deleted
  Compute $a_1, b_1 \in \mathbb{Z}$ such that $a_1 H(x) + b_1 H(y_1) = \gcd(H(x), H(y_1))$
  Compute $\mathtt{w}' \leftarrow (\mathsf{A}')^{a_1} \mathtt{w}^{b_1} \mod n$, and $\mathbf{s}' \leftarrow \mathbf{s} \| (\gcd(H(x), H(y_1)))$

  To verify, parse $\mathbf{s}'$ as $\gcd(H(x), H(y_1))$, compute $\mathtt{x} \leftarrow \frac{H(x)}{\gcd(H(x),H(y_1))}$. Finally, check

$$(\mathtt{w}')^{\mathtt{x}} \stackrel{?}{=} \mathsf{A}'$$

- y_2 was deleted
  Compute $a_2, b_2 \in \mathbb{Z}$ such that $a_2 \mathtt{x} + b_2 H(y_1) = \gcd(\mathtt{x}, H(y_2))$
  Compute $\mathtt{w}'' \leftarrow (\mathsf{A}'')^{a_2} \mathtt{w}^{b_2} \mod n$, and $\mathbf{s}'' \leftarrow \mathbf{s}' \| (\gcd(\mathtt{x}, H(y_2)))$

// Note: $P^+(H(x)) = P^+(\mathtt{x})$

Our Universal Accumulator Construction

New Membership update algorithm:

- $\mathsf{MemWitUp}(\mathsf{pp}, x, w_x, \mathsf{upmsg})$:
  1. Parse pp as (n, u), w_x as $(\mathtt{w}, \mathbf{s})$, and upmsg as $(\mathsf{op}, H(y), \delta, \mathsf{A}, \mathsf{A}')$.
  2. If $\mathsf{op} = \mathsf{add}$, compute $\mathtt{w}' \leftarrow \mathtt{w}^{H(y)} \bmod n$, and let $w'_x = (\mathtt{w}', \mathbf{s})$.
  3. Else if $\mathsf{op} = \mathsf{del}$, do:
     1. Compute $\mathtt{x} \leftarrow H(x) / \prod_{i=1}^{|\mathbf{s}|} \mathbf{s}[i]$.
     2. Compute $a, b \in \mathbb{Z}$ such that $a\mathtt{x} + bH(y) = \gcd(\mathtt{x}, H(y))$.
     3. Compute $\mathtt{w}' \leftarrow (\mathsf{A}')^a \mathtt{w}^b \bmod n$.
     4. If $\gcd(\mathtt{x}, H(y)) \neq 1$, let $\mathbf{s}' \leftarrow \mathbf{s} \| (\gcd(\mathtt{x}, H(y)))$. Otherwise, let $\mathbf{s}' \leftarrow \mathbf{s}$.
     5. Let $w'_x = (\mathtt{w}', \mathbf{s}')$.
  4. Return w'_x.

Our Universal Accumulator Construction

New Membership update algorithm:

- $\mathsf{MemWitUp}(\mathsf{pp}, x, w_x, \mathsf{upmsg})$:
  1. Parse pp as (n, u), w_x as $(\mathtt{w}, \mathbf{s})$, and upmsg as $(\mathsf{op}, H(y), \delta, \mathsf{A}, \mathsf{A}')$.
  2. If $\mathsf{op} = \mathsf{add}$, compute $\mathtt{w}' \leftarrow \mathtt{w}^{H(y)} \bmod n$, and let $w'_x = (\mathtt{w}', \mathbf{s})$.
  3. Else if $\mathsf{op} = \mathsf{del}$, do:
     1. Compute $\mathtt{x} \leftarrow H(x) / \prod_{i=1}^{|\mathbf{s}|} \mathbf{s}[i]$.
     2. Compute $a, b \in \mathbb{Z}$ such that $a\mathtt{x} + bH(y) = \gcd(\mathtt{x}, H(y))$.
     3. Compute $\mathtt{w}' \leftarrow (\mathsf{A}')^a \mathtt{w}^b \bmod n$.
     4. If $\gcd(\mathtt{x}, H(y)) \neq 1$, let $\mathbf{s}' \leftarrow \mathbf{s} \| (\gcd(\mathtt{x}, H(y)))$. Otherwise, let $\mathbf{s}' \leftarrow \mathbf{s}$.
     5. Let $w'_x = (\mathtt{w}', \mathbf{s}')$.
  4. Return w'_x.

$\mathbf{s}$ is a tuple that contains **small/smooth** factors of $H(x)$ and $|\mathbf{s}| \leq \ell$, where ℓ is the output length of H.

Our Universal Accumulator Construction

//$P^+(H(x)) > 2^{\sqrt[4]{\ell}}$ with overwhelming probability.

- $\mathsf{MemVerify}(\mathsf{pp}, \mathsf{A}, x, w_x)$:
  1. Parse pp as (n, u), and w_x as $(\mathtt{w}, \mathbf{s})$.
  2. For $i \in [|\mathbf{s}|]$, if $\mathbf{s}[i] > 2^{\sqrt[4]{\ell}}$, return 0.
  3. Compute $\mathtt{x} \leftarrow H(x) / \prod_{i=1}^{|\mathbf{s}|} \mathbf{s}[i]$.
  4. If $\mathtt{w}^{\mathtt{x}} \equiv \mathsf{A} \mod n$ return 1. Otherwise, return 0.

Line 2 ensures that $P^+(H(x)) = P^+(\mathtt{x})$.

Our Universal Accumulator Construction–Analysis

Definition (Correctness–Informal)

An accumulator scheme is correct if given A_t, $(x, w_{x,t})$, and $(y, \bar{w}_{y,t})$ such that $w_{x,t}, \bar{w}_{y,t}$ are up-to-date:

- $(x, w_{x,t})$ pass MemVerify with overwhelming probability
- $(y, \bar{w}_{y,t})$ pass NonMemVerify with overwhelming probability

Definition (Security–Informal)

An accumulator scheme is secure if for all poly-time adversary $\mathcal{A}$:

- It is *hard* to output a valid w_x for any $x \notin \mathcal{S}$
- It is *hard* to output a valid $\bar{w}_y$ for any $y \in \mathcal{S}$

Our Universal Accumulator Construction–Analysis

Definition (Correctness–Informal)

An accumulator scheme is correct if given A_t, $(x, w_{x,t})$, and $(y, \bar{w}_{y,t})$ such that $w_{x,t}, \bar{w}_{y,t}$ are up-to-date:

- $(x, w_{x,t})$ **pass** MemVerify **with overwhelming probability** ✓
- $(y, \bar{w}_{y,t})$ pass NonMemVerify with overwhelming probability

Definition (Security–Informal)

An accumulator scheme is secure if for all poly-time adversary $\mathcal{A}$:

- It is *hard* to output a valid w_x for any $x \notin \mathcal{S}$
- It is *hard* to output a valid $\bar{w}_y$ for any $y \in \mathcal{S}$

Our Universal Accumulator Construction–Analysis

Definition (Correctness–Informal)

An accumulator scheme is correct if given A_t, $(x, w_{x,t})$, and $(y, \bar{w}_{y,t})$ such that $w_{x,t}, \bar{w}_{y,t}$ are up-to-date:

- $(x, w_{x,t})$ **pass** MemVerify **with overwhelming probability** ✓
- $(y, \bar{w}_{y,t})$ pass NonMemVerify with overwhelming probability

Definition (Security–Informal)

An accumulator scheme is secure if for all poly-time adversary $\mathcal{A}$:

- **It is *hard* to output a valid w_x for any $x \notin \mathcal{S}$** ★
- It is *hard* to output a valid $\bar{w}_y$ for any $y \in \mathcal{S}$

Our Universal Accumulator Construction–Analysis

Security

Definition (Strong RSA Assumption)

Given (u, n), with $u \in \mathbb{Z}_n^*$, output $(v, e) \in \mathbb{Z}_n^* \times \mathbb{Z}$ such that

$$v^e \equiv u \bmod n \quad \wedge \quad e > 1$$

Our Universal Accumulator Construction–Analysis

Security

Suppose adversary $\mathcal{A}$ outputs $\{x_1, \ldots, x_m\}$, $\mathsf{A} = u^{\prod_{i=1}^{m} H(x_i)}$, (x^*, w_{x^*}) such that $x^* \notin \{x_1, \ldots, x_m\}$.

Our Universal Accumulator Construction–Analysis

Security

Suppose adversary $\mathcal{A}$ outputs $\{x_1, \ldots, x_m\}$, $\mathsf{A} = u^{\prod_{i=1}^{m} H(x_i)}$, (x^*, w_{x^*}) such that $x^* \notin \{x_1, \ldots, x_m\}$.

1. $H(x^*) \nmid \prod_{i=1}^{m} H(x_i)$ with overwhelming probability. //Corollary from $P^+(H(x)) > 2^{\sqrt[4]{\ell}}$ w.o.p
2. $w_{x^*} = (\mathtt{w}, \mathbf{s})$ is valid. Therefore, all components of $\mathbf{s}$ are $2^{\sqrt[4]{\ell}}$-smooth.

Our Universal Accumulator Construction–Analysis

Security

Suppose adversary $\mathcal{A}$ outputs $\{x_1, \ldots, x_m\}$, $A = u^{\prod_{i=1}^{m} H(x_i)}$, (x^*, w_{x^*}) such that $x^* \notin \{x_1, \ldots, x_m\}$.

1. $H(x^*) \nmid \prod_{i=1}^{m} H(x_i)$ with overwhelming probability. //Corollary from $P^+(H(x)) > 2^{\sqrt[4]{\ell}}$ w.o.p
2. $w_{x^*} = (\mathtt{w}, \mathbf{s})$ is valid. Therefore, all components of **s** are $2^{\sqrt[4]{\ell}}$-smooth.

   Let $\mathtt{x}^* \leftarrow \frac{H(x^*)}{\prod_{i=1}^{m} \mathbf{s}[i]}$. $P^+(\mathtt{x}^*) = P^+(H(x^*))$, so $\mathtt{x}^* \nmid \prod_{i=1}^{m} H(x_i)$.

Our Universal Accumulator Construction–Analysis

Security

Suppose adversary $\mathcal{A}$ outputs $\{x_1, \ldots, x_m\}$, $A = u^{\prod_{i=1}^{m} H(x_i)}$, (x^*, w_{x^*}) such that $x^* \notin \{x_1, \ldots, x_m\}$.

1. $H(x^*) \nmid \prod_{i=1}^{m} H(x_i)$ with overwhelming probability. //Corollary from $P^+(H(x)) > 2^{\sqrt[4]{\ell}}$ w.o.p
2. $w_{x^*} = (\mathtt{w}, \mathbf{s})$ is valid. Therefore, all components of $\mathbf{s}$ are $2^{\sqrt[4]{\ell}}$-smooth.

   Let $\mathtt{x}^* \leftarrow \frac{H(x^*)}{\prod_{i=1}^{m} \mathbf{s}[i]}$. $P^+(\mathtt{x}^*) = P^+(H(x^*))$, so $\mathtt{x}^* \nmid \prod_{i=1}^{m} H(x_i)$.

Our Universal Accumulator Construction–Analysis

Security

Suppose adversary $\mathcal{A}$ outputs $\{x_1, \ldots, x_m\}$, $\mathrm{A} = u^{\prod_{i=1}^{m} H(x_i)}$, (x^*, w_{x^*}) such that $x^* \notin \{x_1, \ldots, x_m\}$.

1. $H(x^*) \nmid \prod_{i=1}^{m} H(x_i)$ with overwhelming probability. //Corollary from $P^+(H(x)) > 2^{\sqrt[4]{\ell}}$ w.o.p
2. $w_{x^*} = (\mathtt{w}, \mathbf{s})$ is valid. Therefore, all components of $\mathbf{s}$ are $2^{\sqrt[4]{\ell}}$-smooth.

   Let $\mathtt{x}^* \leftarrow \frac{H(x^*)}{\prod_{i=1}^{m} \mathbf{s}[i]}$. $P^+(\mathtt{x}^*) = P^+(H(x^*))$, so $\mathtt{x}^* \nmid \prod_{i=1}^{m} H(x_i)$. Let $\theta = \prod_{i=1}^{m} H(x_i)$

$$\mathtt{w}^{\mathtt{x}^*} = \mathrm{A} = u^{\theta} \qquad (\clubsuit)$$

Our Universal Accumulator Construction–Analysis

Security

Suppose adversary $\mathcal{A}$ outputs $\{x_1, \ldots, x_m\}$, $\mathsf{A} = u^{\prod_{i=1}^{m} H(x_i)}$, (x^*, w_{x^*}) such that $x^* \notin \{x_1, \ldots, x_m\}$.

1. $H(x^*) \nmid \prod_{i=1}^{m} H(x_i)$ with overwhelming probability. //Corollary from $P^+(H(x)) > 2^{\sqrt[4]{\ell}}$ w.o.p
2. $w_{x^*} = (\mathtt{w}, \mathbf{s})$ is valid. Therefore, all components of $\mathbf{s}$ are $2^{\sqrt[4]{\ell}}$-smooth.

Let $\mathtt{x}^* \leftarrow \frac{H(x^*)}{\prod_{i=1}^{m} \mathbf{s}[i]}$. $P^+(\mathtt{x}^*) = P^+(H(x^*))$, so $\mathtt{x}^* \nmid \prod_{i=1}^{m} H(x_i)$. Let $\theta = \prod_{i=1}^{m} H(x_i)$

$$\mathtt{w}^{\mathtt{x}^*} = \mathsf{A} = u^{\theta} \qquad (\clubsuit)$$

Let $\tilde{\mathtt{x}} \leftarrow \frac{\mathtt{x}^*}{\gcd(\mathtt{x}^*, \theta)}$ and $\tilde{\theta} \leftarrow \frac{\theta}{\gcd(\mathtt{x}^*, \theta)}$. Note that $\gcd(\tilde{\mathtt{x}}, \tilde{\theta}) = 1$ and From equation $(\clubsuit)$, $\mathtt{w}^{\tilde{\mathtt{x}}} = u^{\tilde{\theta}}$

Our Universal Accumulator Construction–Analysis

Security

Suppose adversary $\mathcal{A}$ outputs $\{x_1, \ldots, x_m\}$, $\mathsf{A} = u^{\prod_{i=1}^{m} H(x_i)}$, (x^*, w_{x^*}) such that $x^* \notin \{x_1, \ldots, x_m\}$.

1. $H(x^*) \nmid \prod_{i=1}^{m} H(x_i)$ with overwhelming probability. //Corollary from $P^+(H(x)) > 2^{\sqrt[4]{\ell}}$ w.o.p
2. $w_{x^*} = (\mathtt{w}, \mathbf{s})$ is valid. Therefore, all components of $\mathbf{s}$ are $2^{\sqrt[4]{\ell}}$-smooth.

Let $\mathtt{x}^* \leftarrow \frac{H(x^*)}{\prod_{i=1}^{m} \mathbf{s}[i]}$. $P^+(\mathtt{x}^*) = P^+(H(x^*))$, so $\mathtt{x}^* \nmid \prod_{i=1}^{m} H(x_i)$. Let $\theta = \prod_{i=1}^{m} H(x_i)$

$$\mathtt{w}^{\mathtt{x}^*} = \mathsf{A} = u^{\theta} \qquad (\clubsuit)$$

Let $\tilde{\mathtt{x}} \leftarrow \frac{\mathtt{x}^*}{\gcd(\mathtt{x}^*, \theta)}$ and $\tilde{\theta} \leftarrow \frac{\theta}{\gcd(\mathtt{x}^*, \theta)}$. Note that $\gcd(\tilde{\mathtt{x}}, \tilde{\theta}) = 1$ and From equation $(\clubsuit)$, $\mathtt{w}^{\tilde{\mathtt{x}}} = u^{\tilde{\theta}}$
Let $a, b \in \mathbb{Z}$ s.t. $a\tilde{\mathtt{x}} + b\tilde{\theta} = 1$. By Shamir's trick,

$$(u^a \mathtt{w}^b)^{\tilde{\mathtt{x}}} = u^{a\tilde{\mathtt{x}}} u^{b\tilde{\theta}} = u$$

Our Universal Accumulator Construction–Analysis

Security

Suppose adversary $\mathcal{A}$ outputs $\{x_1, \ldots, x_m\}$, $\mathrm{A} = u^{\prod_{i=1}^{m} H(x_i)}$, (x^*, w_{x^*}) such that $x^* \notin \{x_1, \ldots, x_m\}$.

1. $H(x^*) \nmid \prod_{i=1}^{m} H(x_i)$ with overwhelming probability. //Corollary from $P^+(H(x)) > 2^{\sqrt[4]{\ell}}$ w.o.p
2. $w_{x^*} = (\mathtt{w}, \mathbf{s})$ is valid. Therefore, all components of $\mathbf{s}$ are $2^{\sqrt[4]{\ell}}$-smooth.

Let $\mathtt{x}^* \leftarrow \frac{H(x^*)}{\prod_{i=1}^{m} \mathbf{s}[i]}$. $P^+(\mathtt{x}^*) = P^+(H(x^*))$, so $\mathtt{x}^* \nmid \prod_{i=1}^{m} H(x_i)$. Let $\theta = \prod_{i=1}^{m} H(x_i)$

$$\mathtt{w}^{\mathtt{x}^*} = \mathrm{A} = u^{\theta} \qquad (\clubsuit)$$

Let $\tilde{\mathtt{x}} \leftarrow \frac{\mathtt{x}^*}{\gcd(\mathtt{x}^*, \theta)}$ and $\tilde{\theta} \leftarrow \frac{\theta}{\gcd(\mathtt{x}^*, \theta)}$. Note that $\gcd(\tilde{\mathtt{x}}, \tilde{\theta}) = 1$ and From equation $(\clubsuit)$, $\mathtt{w}^{\tilde{\mathtt{x}}} = u^{\tilde{\theta}}$
Let $a, b \in \mathbb{Z}$ s.t. $a\tilde{\mathtt{x}} + b\tilde{\theta} = 1$. By Shamir's trick,

$$(u^a \mathtt{w}^b)^{\tilde{\mathtt{x}}} = u^{a\tilde{\mathtt{x}}} u^{b\tilde{\theta}} = u$$

Therefore $u^a \mathtt{w}^b$ is an $\tilde{\mathtt{x}}$-root of u

Some Experimental results

λ	H_{Prime} length (bit)	H_{Prime} time (ms)	H_{Odd} length (bit)	H_{Odd} time (ms)
112	232	10.65	1440	0.48
128	264	13.62	1704	0.60
192	393	31.9	2896	1.07
256	521	52.31	4208	1.56

Table: H_{Prime} versus H_{Odd}.

Some Experimental results

λ	$\mathsf{Add}^{(H_{\mathsf{Prime}},\mathsf{sk})}$ time (ms)	$\mathsf{Add}^{(H_{\mathsf{Odd}},\mathsf{sk})}$ time (ms)	$\mathsf{Add}^{H_{\mathsf{Prime}}}$ time (ms)	$\mathsf{Add}^{H_{\mathsf{Odd}}}$ time (ms)
112	12.83	2.73	11.06	1.96
128	20.37	7.38	14.27	3.98
192	99.48	68.84	35.34	25.10
256	456.10	402.71	65.97	110.6

Table: Comparison of different Add algorithms. $\mathsf{Add}^{(H,\mathsf{sk})}$ represents the addition procedure that uses the secret key sk and H as the underlying hash function, and Add^{H} represents the addition procedure that is performed without sk using H as the underlying hash function.

Some Experimental results

λ	$\text{Add}^{(H_{\text{Prime}},\text{sk})}$ time (ms)	$\text{Add}^{(H_{\text{Odd}},\text{sk})}$ time (ms)	$\text{Add}^{H_{\text{Prime}}}$ time (ms)	$\text{Add}^{H_{\text{Odd}}}$ time (ms)
112	12.83	2.73	11.06	1.96
128	20.37	7.38	14.27	3.98
192	99.48	68.84	35.34	25.10
256	456.10	402.71	65.97	110.6

Table: Comparison of different Add algorithms. $\text{Add}^{(H,\text{sk})}$ represents the addition procedure that uses the secret key sk and H as the underlying hash function, and Add^{H} represents the addition procedure that is performed without sk using H as the underlying hash function.

Outline

1. **Universal Accumulator:** Gen, Add, Delete, MemVerify, MemWitUp ✓
2. Wesolowski's Proof of Exponentation without hashing to primes, i.e., SimPoE

Proof of Exponentation

Definition

A Proof of Exponentation (PoE) is an interactive protocol (argument) for the language

$$\mathcal{L}_{\mathsf{PoE},\mathbb{G}} = \{(v, u, e) \in \mathbb{G}^2 \times \mathbb{Z} : v^e = u\}$$

where $\mathbb{G}$ is a group of unknown order.

Proof of Exponentation

Definition

A Proof of Exponentation (PoE) is an interactive protocol (argument) for the language

$$\mathcal{L}_{\mathsf{PoE},\mathbb{G}} = \{(v, u, e) \in \mathbb{G}^2 \times \mathbb{Z} : v^e = u\}$$

where $\mathbb{G}$ is a group of unknown order.

We recall Wesolowski's PoE [Wes20]:

Initialization:

1. Sample and output a group $\mathbb{G}$ of unknown order.
2. **Statement:** $(v, u, e) \in \mathbb{G}^2 \times \mathbb{Z}$.

Interaction:

1. V samples $c \leftarrow\$ \ \mathtt{PRIMES}(2^\lambda)$ and sends it to P.
2. P computes $\pi \leftarrow v^{\lfloor e/c \rfloor}$ and sends it to V.
3. V computes $r \leftarrow e \mod c$. Then, it outputs 1 if $\pi^c v^r = u$. Otherwise, it outputs 0.

Proof of Exponentation : SimPoE

Definition

A Proof of Exponentation (PoE) is an interactive protocol (argument) for the language

$$\mathcal{L}_{\mathsf{PoE},\mathbb{G}} = \{(v, u, e) \in \mathbb{G}^2 \times \mathbb{Z} : v^e = u\}$$

where $\mathbb{G}$ is a group of unknown order.

We present *SimPoE*:

Initialization:

1. Sample and output a group $\mathbb{G}$ of unknown order.
2. **Statement:** $(v, u, e) \in \mathbb{G}^2 \times \mathbb{Z}$**.**

Interaction:

1. V samples $c \leftarrow\$ \mathsf{Odds}(2^{\ell-1}, 2^{\ell} - 1)$ and sends it to P.
2. P computes $\pi \leftarrow v^{\lfloor e/c \rfloor}$ and sends it to V.
3. V computes $r \leftarrow e \mod c$. Then, it outputs 1 if $\pi^c v^r = u$. Otherwise, it outputs 0.

Thank You

https://ia.cr/2024/505