Seminar Report

Topic 8: Dynamic and Universal Accumulators (Strong RSA Assumption)

Seminar: Cryptography and Data Security,
University of Bern, Spring 2025

Vishrut Shukla
University of Fribourg
vishrut.shukla@unifr.ch

30 May 2025

1 Overview

Accumulators were first introduced by Benaloh and de Mare [BM94] as a method to condense a set of values into one short accumulator, such that there is a short witness for each value that has been accumulated, while it is infeasible to find a witness for a value that has not been accumulated. This primitive enables efficient membership testing, time stamping, and authenticated directory services.

Barić and Pfitzmann [BP97] proposed a collision-resistant accumulator under the strong RSA assumption, which improved the security guarantees of the original construction. This work established the foundation for practical accumulator implementations.

Camenisch and Lysyanskaya [CL02] introduced dynamic accumulators that enable efficient updates to the accumulated set. The key innovation was that elements could be added or removed with constant computational cost, independent of the size of the accumulated set. This made accumulators practical for applications requiring frequent modifications, such as certificate revocation lists.

A *universal accumulator* [LLX07] extends this concept by providing both membership and non-membership witnesses for any queried element. The ”universal” property comes from the fact that each possible value in the input domain has a witness - either a membership witness (for values in the set) or a non-membership witness (for values not in the set).

The scheme achieves three key properties:

- **Compactness:** The accumulator c is a single group element, independent of $|X|$, where $|X|$ denotes the number of elements in the accumulated set X.
- **Efficient Proofs:** Both membership and non-membership witnesses can be verified with constant modular exponentiations.
- **Dynamic Updates:** Elements can be added or removed with a single exponentiation, and witnesses can be updated without full re-accumulation.

Universal accumulators have been applied to certificate revocation lists and privacy-preserving credential systems [LLX07]. The concept of stateless blockchains, introduced by Boneh et al. [BBF19], leverages these properties to enable nodes to validate transactions without storing the entire state.

The rest of this report will discuss the algorithms and constructions of universal accumulators, including their security properties, implementation details, and applications.

2 Mathematical Background

This section introduces the notation and concepts used throughout the report.

Notation.

- For a positive integer k, we write 1^k for the unary representation.
- $\{0,1\}^k$ denotes the set of all bit-strings of length k. We write $\{0,1\}^*$ for all finite bit-strings.
- If X is a set, then $x \leftarrow X$ means "draw x uniformly at random from X."
- A function $\nu : \mathbb{N} \to \mathbb{R}$ is *negligible* if for every constant $c > 0$, $\nu(k) < k^{-c}$ for all sufficiently large k.

Asymptotic Complexity.

- A function $\nu : \mathbb{N} \to \mathbb{R}$ is *negligible* if for every constant $c > 0$, $\nu(k) < k^{-c}$ for all sufficiently large k. We use negligible functions to bound the success probability of adversaries in security proofs.
- A *probabilistic polynomial-time* (PPT) algorithm is one that runs in time polynomial in its input length and can make random choices. We use PPT to model computationally bounded adversaries in our security analysis.

Prime Numbers and Safe Primes.

- A *prime* is an integer having exactly two divisors.
- A *safe prime* is of the form $p = 2p' + 1$ where p' is also prime.
- Pollard's $p-1$ algorithm (1974) factors $n = pq$ if $p-1$ or $q-1$ is B-smooth, that is all prime-factors of $p-1$ or $q-1$ are smaller than a bound B. The algorithm will then factor n in time $\mathcal{O}(B \log B \log^2 n)$. To resist this, both $p-1$ and $q-1$ should each have at least one large prime factor (e.g. $> 2^{80}$ bits).

Modular Arithmetic and Groups.

- For a modulus $n \geq 2$, $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ with addition and multiplication modulo n.
- The multiplicative group of units is $\mathbb{Z}_n^* = \{\, g \in \mathbb{Z}_n : \gcd(g, n) = 1\}$, where gcd is the greatest common divisor.
- The group of quadratic residues modulo n is

$$QR_n = \{\, y^2 \bmod n : y \in \mathbb{Z}_n^*\}.$$

- Euler's totient function $\phi(n)$ counts the number of integers between 1 and n that are coprime to n. For an RSA modulus $n = pq$ where p and q are distinct primes, $\phi(n) = (p-1)(q-1)$. This value represents the order of the multiplicative group $\mathbb{Z}_n^*$, meaning there are exactly $\phi(n)$ elements in $\mathbb{Z}_n^*$.

Extended Euclidean Algorithm. For integers a, b, the algorithm finds $u, v \in \mathbb{Z}$ such that $u\,a + v\,b = \gcd(a, b)$. When $\gcd(a, b) = 1$, this yields Bézout coefficients satisfying $u\,a + v\,b = 1$.

Lemma 1 (Efficient Root Extraction). *Let n be any integer where $u, v \in \mathbb{Z}_n^*$ and $a, b \in \mathbb{Z}$ satisfy:*

$$u^a \equiv v^b \pmod{n} \quad \textit{with} \quad \gcd(a, b) = 1.$$

Then $\exists\, x \in \mathbb{Z}_n^*$ computable in polynomial time such that $x^a \equiv v \pmod{n}$.

Proof. By Bézout's identity, $\exists\, c, d \in \mathbb{Z}$ with $bd = 1 + ac$. Compute:

$$x := u^d v^{-c} \bmod n \implies x^a = (u^d v^{-c})^a = u^{ad} v^{-ac} = (u^a)^d (v^b)^{-c} = v^{bd-bc} = v^1 = v \pmod{n}.$$

□

3 Security Assumption

The security of RSA accumulators is based on the strong RSA assumption, which assumes that it is infeasible to solve the following problem: Given an RSA modulus n and a random $x \leftarrow_R \mathbb{Z}_n^*$, find $e > 1$ and $y \in \mathbb{Z}_n^*$ such that $y^e \equiv x \mod n$

Assumption 1 (Strong RSA Assumption). For every probabilistic polynomial-time algorithm A,

$$\Pr\left[n \leftarrow G(1^k), x \leftarrow_R \mathbb{Z}_n, (y, e) \leftarrow A(n, x) : y^e \equiv x \pmod{n} \wedge 1 < e < n\right] = \text{neg}(k)$$

where $G(1^k)$ is an algorithm that generates an RSA modulus n of size k, and $\text{neg}(k)$ is a negligible function.

4 Cryptographic Notion

This section states what the accumulator scheme achieves without describing its internal workings.

Syntax and Algorithms. A universal accumulator scheme consists of six polynomial-time algorithms. The setup algorithm $\mathsf{Setup}(1^k)$ generates an RSA modulus $n = pq$ and trapdoor information $\mathsf{aux}_n = (p, q)$. The core accumulation is performed by $\mathsf{Acc}(n, g, X)$, which takes modulus n, base element $g \in QR_n$, and set $X \subset X_k$ to produce accumulator value $c = g^{\prod_{x \in X} x} \bmod n$.

For membership proofs, $\mathsf{GenWitness}(n, g, X, x)$ generates a witness w_1 for $x \in X$, verified by $\mathsf{VerMem}(n, c, x, w_1)$. Similarly, $\mathsf{GenNonWitness}(n, g, X, x)$ produces a witness w_2 for $x \notin X$, verified by $\mathsf{VerNonMem}(n, c, x, w_2)$.

Quasi-commutativity. The accumulator is quasi-commutative: for all k, for all $f \in \mathcal{F}_k$, for all $u \in \mathcal{U}_f$, and for all $x_1, x_2 \in X_k$,

$$f(f(u, x_1), x_2) = f(f(u, x_2), x_1)$$

This property ensures that the order of element insertion does not affect the final accumulator value. If $X = \{x_1, \ldots, x_m\} \subset X_k$, then by $f(u, X)$ we denote $f(f(\ldots(f(u, x_1), \ldots), x_m)$. The quasi-commutative property is crucial for efficiency: it allows witness updates to be performed in constant time, independent of the size of the accumulated set, as witnesses can be updated by considering only the changes to the set rather than recomputing from scratch.

Correctness (Completeness). The scheme satisfies perfect correctness: for every security parameter k, key pair $(n, \mathsf{aux}_n) \leftarrow \mathsf{Setup}(1^k)$, generator $g \in QR_n$, set $X \subseteq X_k$, and element $x \in X_k$:

$$\mathsf{VerMem}\big(n, \mathsf{Acc}(n, g, X), x, \mathsf{GenWitness}(n, g, X, x)\big) = 1 \text{ when } x \in X,$$

$$\mathsf{VerNonMem}\big(n, \mathsf{Acc}(n, g, X), x, \mathsf{GenNonWitness}(n, g, X, x)\big) = 1 \text{ when } x \notin X.$$

Security. The fundamental security property ensures that no efficient adversary can produce contradictory proofs. No PPT algorithm can output (X, x, w_1, w_2) such that both $\mathsf{VerMem}(n, \mathsf{Acc}(n, g, X), x, w_1) = 1$ and $\mathsf{VerNonMem}(n, \mathsf{Acc}(n, g, X), x, w_2) = 1$ hold simultaneously, except with negligible probability.

Intuition. The accumulator $c = g^{\prod_{x \in X} x} \bmod n$ compactly encodes X. A membership witness shows how to remove one factor from the exponent, proving inclusion. A non-membership witness uses Bézout coefficients to show the element cannot divide the exponent, proving exclusion. Security ensures no element admits both proofs.

Dynamic Extensions. The accumulator achieves dynamicity through efficient update operations. When elements are added or removed, both the accumulator value and existing witnesses can be updated incrementally without full recomputation.

For updating the accumulator, algorithm D computes $\mathsf{D}(c, \hat{x})$ to obtain either $\mathsf{Acc}(n, g, X \cup \{\hat{x}\})$ or $\mathsf{Acc}(n, g, X \setminus \{\hat{x}\})$. Witness updates are handled by algorithms W_1 and W_2 for membership and non-membership witnesses respectively, ensuring that witnesses remain valid under the updated accumulator.

5 Implementation

This section describes the concrete algorithms for the universal accumulator and its dynamic operations. Let k be a security parameter and $l = k/2 - 2$. The input domain X_k consists of all primes in $\mathbb{Z}_{2^l}$, and $\mathcal{F}_k$ represents the family of functions corresponding to safe-prime products of length l.

The accumulator scheme begins with a setup phase that generates the necessary cryptographic parameters. The **Setup** algorithm takes the security parameter 1^k as input and generates safe primes $p = 2p' + 1$ and $q = 2q' + 1$ of $\frac{k}{2} - 1$ bits. It computes the RSA modulus $n = pq$ and stores the auxiliary information $\mathsf{aux}_n = (p, q)$. A random generator $g \in QR_n$ is selected, and an empty set X is initialized. The algorithm returns the public parameters $(n, g, \mathsf{aux}_n, X)$.

The core accumulation operation is performed by the **Accumulate** algorithm, which takes the public key (n, g), an element $x \in X_k$, and the current set X as input. It first verifies that x belongs to the input domain X_k. If valid, it adds x to the set X and computes the new accumulator value $c = g^x \bmod n$. The algorithm then generates a witness for x using **GenWitness** and updates all existing witnesses using **UpdateMem**. The final output consists of the new accumulator value and the witness.

To remove elements from the accumulator, the **Delete** algorithm takes the current accumulator value c, the element x to be removed, the set X, and the auxiliary information aux_n. It verifies that x is present in X before proceeding. Upon successful verification, it removes x from X and computes the new accumulator value $\hat{c} = c^{x^{-1} \bmod \phi(n)} \bmod n$. The algorithm then updates all witnesses using **UpdateMem** and returns the new accumulator value.

Witness generation and verification are handled by four algorithms. The **GenWitness** algorithm generates a membership witness for an element $x \in X$ by computing $w = g^{\prod_{y \in X \setminus \{x\}} y} \bmod n$. This witness can be verified using **VerifyWitness**, which checks if $w^x \equiv c \bmod n$. For non-membership proofs, **GenNonWitness** computes a witness (a, d) for an element $x \notin X$ by finding Bézout coefficients (a, b) such that $a \prod_{y \in X} y + b\, x = 1$ and setting $d = g^{-b} \bmod n$. The **VerifyNonWitness** algorithm verifies non-membership by checking if $c^a \equiv d^x\, g \bmod n$.

The accumulator supports dynamic updates through two algorithms. The **UpdateMem** algorithm updates all membership witnesses when the accumulator changes. For each element $y \in X$, it computes a new witness either by exponentiating the old witness (for insertions) or using Bézout coefficients (for deletions). Similarly, **UpdateNonMem** updates all non-membership witnesses by computing new coefficients and adjusting the witness components accordingly.

6 Security Argument

Theorem 1 (Security of Universal Accumulator). *Under the Strong RSA assumption, the above construction is a secure universal accumulator.*

Proof. We assume all arithmetic operations in this proof are modulo n unless specified otherwise.

Suppose there exists a polynomial-time adversary $\mathcal{A}$ which, on input n and $g \in QR_n$, outputs $c_x \in \mathcal{G}_f$, $d \in \mathcal{G}_f$, $x \in X_k$, $a \in \mathbb{Z}_{2^l}$, and $X = \{x_1, \ldots, x_m\} \subset X_k$, such that:

$$c = g^{x_1 \cdots x_m}, \tag{1}$$

$$(c_x)^x = c, \tag{2}$$

$$c^a = d^x \cdot g. \tag{3}$$

We can construct an algorithm $\mathcal{B}$ to break the Strong RSA assumption by invoking $\mathcal{A}$. Let $u = \prod_{i=1}^{m} x_i$. We consider two cases:

- **Case 1:** $x \in X$. The adversary computes u, a, d, and x such that $c = g^u$ and $c^a = d^x \cdot g$. That is, the adversary computes u, a, d, and x such that $g^{au-1} = d^x$. Because $x \in X$, $x \mid u$, and $\gcd(au - 1, x) = 1$. By Lemma 1, we can efficiently find y such that $y^x = g$.
- **Case 2:** $x \notin X$. The adversary computes u, c_x, and x such that $c = g^u$ and $(c_x)^x = c$. In other words, the adversary can find u, c_x, and x such that $g^u = (c_x)^x$. As $x_1, \ldots, x_m$ are all prime, and $x \notin X$, clearly $\gcd(x, u) = 1$. By Lemma 1, we can efficiently find y such that $y^x = g$.

We now construct an efficient algorithm $\mathcal{B}$ that breaks the Strong RSA assumption as follows. Given an RSA modulus n and $g \leftarrow QR_n$, $\mathcal{B}$ invokes $\mathcal{A}$ with input n and g, and obtains outputs c_x, d, x, a, X from $\mathcal{A}$. By the preceding arguments, $\mathcal{B}$ can efficiently compute y from c_x, d, x, a, X such that $y^x = g$, which contradicts the Strong RSA assumption. □

Corollary (Non-membership Witness Validity). For any $f \in \mathcal{F}_k$ and any given set $X \subset X_k$, it is computationally infeasible to find $x \in X$ with a valid non-membership witness.

Proof. This follows directly from Theorem 1. □

References

[BBF19] D. Boneh, B. Bünz, and B. Fisch. "Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains". In: *Advances in Cryptology - CRYPTO 2019*. Vol. 11692. Lecture Notes in Computer Science. Springer, 2019, pp. 561–586. DOI: 10.1007/978-3-030-26948-7_20.

[BM94] J. Benaloh and M. de Mare. "One-Way Accumulators: A Decentralized Alternative to Digital Signatures". In: *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993. Proceedings*. Vol. 765. Lecture Notes in Computer Science. Springer, 1994, pp. 274–285. DOI: 10.1007/3-540-48285-7_24.

[BP97] N. Barić and B. Pfitzmann. "Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees". In: *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*. Vol. 1233. Lecture Notes in Computer Science. Springer, 1997, pp. 480–494. DOI: 10.1007/3-540-69053-0_33.

[CL02] J. Camenisch and A. Lysyanskaya. "Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials". In: *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*. Vol. 2442. Lecture Notes in Computer Science. Springer, 2002, pp. 61–76. DOI: 10.1007/3-540-45708-9_5.

[LLX07] J. Li, N. Li, and R. Xue. "Universal Accumulators with Efficient Nonmembership Proofs". In: *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*. Vol. 4521. Lecture Notes in Computer Science. Springer, 2007, pp. 253–269. DOI: 10.1007/978-3-540-72738-5_17.

A Pseudocode example

The following pseudocode summarizes the main operations of the universal accumulator:

Variable Descriptions:

- k: Security parameter; n: RSA modulus (product of safe primes); g: Generator of quadratic residues modulo n
- X: Set of elements to be accumulated; x: Individual element for which witness is generated
- c: Accumulator value; w: Witness (either membership or non-membership)
- u: Product of all elements in set X; u': Product of all elements in X except x
- (a, b): Bézout coefficients; d: Component of non-membership witness
- $\hat{x}$: New element for update operations; $\hat{c}$: Updated accumulator value; $\hat{w}$: Updated witness
- aux_n: Auxiliary information (prime factors of n); $\perp$: Error indicator for invalid inputs

Algorithm 1 Pseudocode for universal accumulator operations

function $Setup(1^k)$
  generate safe primes p', q' of $\frac{k}{2} - 1$ bits
  set $p = 2p' + 1$, $q = 2q' + 1$, $n = pq$, $\mathsf{aux}_n = (p, q)$
  pick random $g \in QR_n$
  initialize empty set $X \leftarrow \emptyset$
  return $(n, g, \mathsf{aux}_n, X)$

function $Accumulate(n, g, x, X)$
  if $x \notin X_k$ **then return** $\perp$
  add x to set X
  compute $c \leftarrow g^x \bmod n$
  compute $w \leftarrow GenWitness(n, g, X, x)$
  update all witnesses using $UpdateMem(w, c, c, x, x)$
  return (c, w)

function $Delete(n, c, x, X, \mathsf{aux}_n)$
  if $x \notin X$ **then return** $\perp$
  remove x from set X
  compute $\hat{c} \leftarrow c^{x^{-1} \bmod \phi(n)} \bmod n$
  update all witnesses using $UpdateMem(w, c, \hat{c}, x, x)$
  return $\hat{c}$

function $GenWitness(n, g, X, x)$
  let $u' \leftarrow \prod_{y \in X \setminus \{x\}} y$
  compute $w \leftarrow g^{u'} \bmod n$
  return w

function $VerifyWitness(n, c, x, w)$
  if $x \notin X_k$ **then return** 0
  return 1 if $w^x \equiv c \bmod n$, else 0

function $GenNonWitness(n, g, X, x)$
  if $x \notin X_k$ **then return** $\perp$
  if $x \in X$ **then return** $\perp$
  let $u \leftarrow \prod_{y \in X} y$ and find (a, b) with $a\,u + b\,x = 1$
  set $d \leftarrow g^{-b} \bmod n$
  return (a, d)

function $VerifyNonWitness(n, c, x, (a, d))$
  if $x \notin X_k$ **then return** 0
  return 1 if $c^a \equiv d^x\, g \bmod n$, else 0

function $UpdateMem(w, c, \hat{c}, x, \hat{x})$
  for each $y \in X$ **do**
    if $\hat{x} \notin X$ **then** $\hat{w} \leftarrow w^{\hat{x}} \bmod n$
    else find (a, b) with $a\,x + b\,\hat{x} = 1$ and $\hat{w} \leftarrow w^b\, \hat{c}^a \bmod n$
    update witness for y with $\hat{w}$
  return updated witnesses

function $UpdateNonMem(w, c, \hat{c}, x, \hat{x})$
  for each $y \notin X$ **do**
    if $\hat{x} \notin X$ **then** find $(\hat{a}, r)$ with $\hat{a}\,\hat{x} + r\,x = 1$ and $\hat{w} \leftarrow (\hat{a},\ d\,c^r \bmod n)$
    else choose r with $a\,\hat{x} - r\,x \geq 0$ and $\hat{w} \leftarrow (a\,\hat{x} - r\,x,\ d\,\hat{c}^{-r} \bmod n)$
    update witness for y with $\hat{w}$
  return updated witnesses