Zero Knowledge Accumulators and Set Operations

Esha Ghosh\textsuperscript{1} Olya Ohrimenko\textsuperscript{2} Dimitrios Papadopoulos\textsuperscript{3} Roberto Tamassia\textsuperscript{1} Nikos Triandopoulos\textsuperscript{4}

\textsuperscript{1}Brown University \textsuperscript{2}Microsoft Research \textsuperscript{3}University of Maryland \textsuperscript{4}Stevens Institute of Technology

Research supported in part by the US National Science Foundation
Data Outsourcing

OWNER  SERVER

CLIENTS

query

answer
Verifiable data outsourcing (static)

OWNER

CLIENTS

query

answer,

proof

θ

σ

SERVER
Verifiable data outsourcing (dynamic)

OWNER

SERVER

CLIENTS

query
answer,
proof

σ'
θ'

state

Upd

query

answer,
proof
Merkle tree with items stored in sorted order at the leaves.
Proof of $x$: $((a, L), (b, R))$.
Verification: $h(h(a, h(x), c) = r$
Proof leaks rank of item.
Zone enumeration attack

Zone names:
- a.com
- b.com
- w.com

Primary resolver
- pk, \( H(.) \)

Secondary resolver
- \( \theta \)

CLIENTS
- q.com

answer,
proof

\( H(a.com) \)  a5bb  23ce
\( H(b.com) \)  23ce  a5bb
\( H(w.com) \)  fae1  fae1

[qнтересный номер – NDSS15, RFC 5155]
Zone enumeration attack

Primary resolver

Secondary resolver

CLIENTS

q.com

⊥,

23ce

a5bb

θ

pk,H(.)

θ

Zone names:

a.com

b.com

w.com

H(a.com)  a5bb  23ce
H(b.com)  23ce  a5bb
H(w.com)  fae1  fae1

[ Bernstein11–nsec3walker ]
Cryptographic accumulator [Benaloh and del Mare93]

\[ \sigma \leftarrow \text{acc}(\text{Set}\mathcal{X}). \]

**Efficient and succinct proof** for \( x \in \mathcal{X}, x \notin \mathcal{X} \).

**Proofs** are publicly computable and verifiable.

**Soundness**: Forging proof for an element is infeasible.

Traditional proofs are leaky.
In this work

Formal model for zero-knowledge universal dynamic accumulators.

Efficient \textit{construction} for zero-knowledge accumulators.

Efficient \textit{construction} for:

1. is-subset
2. difference
3. union
4. intersection
Our Model

OWNER
CLIENTS
query
answer,
proof

CLIENTS

SERVER

state

σ'
θ'

Upd

query

answer,
proof
Soundness

Challenger

Set $\mathcal{X}_0$

Digests $\sigma_0, \theta_0$

Update $U_i$

$\sigma_{i+1}, \text{Upd}_i$

Repeat $\lambda$ times

Adversary

pk

$j, \text{query}^*, \text{answer}^*, \text{proof}^*$

Figure: Probability that Verify accepts but $\text{answer}^*$ is not correct wrt $\text{query}^*$ on $\mathcal{X}_j$ is negligible
Zero-Knowledge

Challenger  Adversary  Simulator

\[ \text{pk} \quad \xrightarrow{\text{Set } \chi_0} \quad \text{pk} \]

\[ \text{Client } \sigma_0 \quad \xrightarrow{\text{query}} \quad \text{Adversary} \]

\[ \xrightarrow{\text{answer,proof}}, U_i \]

\[ \xleftarrow{\sigma_{i+1}} \]

\[ \text{Guess} \]

\[ \text{Notify Update} \]

\[ \sigma_{i+1} \]

Figure: Probability that Adversary guesses correctly if it is talking to a challenger or a simulator is negligible
Zero Knowledge Accumulator
Client Query: Is element $x \in \mathcal{X}$?

Server Response: answer $= 1$ indication yes and answer $= 0$ indicating no + proof
Set Representation

A set $\mathcal{X} = \{x_1, \ldots, x_N\}$ represented using its characteristic polynomial $\text{Ch}_{\mathcal{X}}[z] = \prod_{i=1}^{N}(z + x_i)$

Bilinear Map:

- $\lambda \in \mathbb{N}$ is the security parameter of the scheme
- $G, G_1$ multiplicative cyclic groups of prime order $p$
- $p$ is a large $k$-bit prime
- $g$ is a random generator of $G$
- $e : G \times G \rightarrow G_1$ is computable bilinear nondegenerate map
- $e(g^a, g^b) = e(g, g)^{ab}$. 
Keygen and Setup (Owner)

\[(sk, pk) \leftarrow \text{KeyGen}(1^\lambda)\]

- Generate bilinear parameters \( pub = (p, G, G_1, e, g) \).
  \( O(\text{poly}(\lambda)) \)
- Choose \( s \leftarrow \mathbb{Z}_p^* \).
- Set \( sk = s \) and \( pk = (g^s, pub) \).

\[(\sigma_0, \theta_0, \text{state}_0) \leftarrow \text{Setup}(sk, \mathcal{X}_0)\]

- Choose \( r \leftarrow \mathbb{Z}_p^* \).
- Set \( \sigma_0 = g^{r \cdot Ch \cdot \mathcal{X}(s)} \).
  \( O(N) \)
- Set \( \theta_0 = (g, g^s, g^{s^2}, \ldots, g^{s^N}, r) \).
  \( O(N) \)
- Set \( \text{state}_0 = \mathcal{X} \).
Query (Server)

\[(answer, \text{proof}) \leftarrow \text{PerformQuery}(\mathcal{X}_j, \theta_j, \text{query})\]

- if \(\text{query} = x \in \mathcal{X}\):
  
  set \(\text{answer} = 1\) and \(\text{proof} = (\sigma_j)^{\frac{1}{s+x}} = g^{\frac{r \cdot \text{Ch}_\mathcal{X}(s)}{(s+x)}}\).  \(O(N \log N)\)
Query (Server)

(\text{answer, proof}) \leftarrow \text{PerformQuery}(\mathcal{X}_j, \theta_j, \text{query})

- if \text{query} = x \in \mathcal{X}:
  
  set \text{answer} = 1 \text{ and } \text{proof} = (\sigma_j)^{\frac{1}{s+x}} = g^{\frac{r \cdot \text{Ch}_\mathcal{X}(s)}{(s+x)}}. \text{O}(N \log N)

- if \text{query} = x \notin \mathcal{X}:
Query (Server)

\[(answer, proof) \leftarrow \text{PerformQuery}(\mathcal{X}_j, \theta_j, \text{query})\]

- if \(\text{query} = x \in \mathcal{X}\):
  
  set \(\text{answer} = 1\) and \(\text{proof} = (\sigma_j) \frac{1}{s+x} g^{r \cdot \text{Ch}_{\mathcal{X}}(s)} (s+x). \text{O}(N \log N)\)

- if \(\text{query} = x \notin \mathcal{X}\):
  1. Using the Extended Euclidean algorithm, compute polynomials \(q_1[z], q_2[z]\) such that \(q_1[z] \text{Ch}_{\mathcal{X}}[z] + q_2[z](z + x) = 1. \text{O}(N \log^2 N \log \log N)\)
Query (Server)

\[(answer, proof) \leftarrow \text{PerformQuery}(X_j, \theta_j, \text{query})\]

- if query = \(x \in X\):
  set answer = 1 and proof = \((\sigma_j) \frac{1}{s+x} = g^{r \cdot Ch X(s)}{(s+x)}. \text{O}(N \log N)\)

- if query = \(x \notin X\):
  1. Using the Extended Euclidean algorithm, compute polynomials \(q_1[z], q_2[z]\) such that \(q_1[z]Ch X[z] + q_2[z](z + x) = 1. \text{O}(N \log^2 N \log \log N)\)
  2. Pick a random \(\gamma \leftarrow Z_p^*\)
Query (Server)

\[(\text{answer, proof}) \leftarrow \text{PerformQuery}(\mathcal{X}_j, \theta_j, \text{query})\]

- if \(\text{query} = x \in \mathcal{X}\):
  
  set \(\text{answer} = 1\) and \(\text{proof} = (\sigma_j)^{\frac{1}{s+x}} = g^{\frac{r \cdot \text{Ch}_{\mathcal{X}}(s)}{(s+x)}}. \ O(N \log N)\)

- if \(\text{query} = x \notin \mathcal{X}\):

  1. Using the Extended Euclidean algorithm, compute polynomials \(q_1[z], q_2[z]\) such that \(q_1[z] \text{Ch}_{\mathcal{X}}[z] + q_2[z](z + x) = 1\). \(O(N \log^2 N \log \log N)\)

  2. Pick a random \(\gamma \leftarrow \mathbb{Z}_p^*\)

  3. Set \(q_1'[z] = q_1[z] + \gamma \cdot (z + x)\)

"
Query (Server)

\[
(\text{answer, proof}) \leftarrow \text{PerformQuery}(\mathcal{X}_j, \theta_j, \text{query})
\]

- if \( \text{query} = x \in \mathcal{X} \):
  
  set \( \text{answer} = 1 \) and \( \text{proof} = (\sigma_j)^{\frac{1}{s+x}} = g^{\frac{r \cdot \text{Ch}_\mathcal{X}(s)}{(s+x)}}. \quad O(N \log N) \)

- if \( \text{query} = x \notin \mathcal{X} \):
  
  1. Using the Extended Euclidean algorithm, compute polynomials \( q_1[z], q_2[z] \) such that \( q_1[z] \text{Ch}_\mathcal{X}[z] + q_2[z](z + x) = 1. \quad O(N \log^2 N \log \log N) \)

  2. Pick a random \( \gamma \leftarrow \mathbb{Z}_p^* \)

  3. Set \( q_1'[z] = q_1[z] + \gamma \cdot (z + x) \)

  4. Set \( q_2'[z] = q_2[z] - \gamma \cdot \text{Ch}_\mathcal{X}[z]. \)
Query (Server)

\[(\text{answer, proof}) \leftarrow \text{PerformQuery}(\mathcal{X}_j, \theta_j, \text{query})\]

- if query = \(x \in \mathcal{X}\):
  
  set answer = 1 and proof = \((\sigma_j)^{\frac{1}{s+x}} = g^{\frac{r \cdot \text{Ch}\mathcal{X}(s)}{(s+x)}} \cdot O(N \log N)\)

- if query = \(x \notin \mathcal{X}\):
  1. Using the Extended Euclidean algorithm, compute polynomials \(q_1[z], q_2[z]\) such that \(q_1[z]\text{Ch}\mathcal{X}[z] + q_2[z](z + x) = 1\).
     \[O(N \log^2 N \log \log N)\]
  2. Pick a random \(\gamma \leftarrow \mathbb{Z}_p^*\)
  3. Set \(q'_1[z] = q_1[z] + \gamma \cdot (z + x)\)
  4. Set \(q'_2[z] = q_2[z] - \gamma \cdot \text{Ch}\mathcal{X}[z]\).
  5. Set \(W_1 := g^{q'_1(s)r^{-1}}, W_2 = g^{q'_2(s)}\).
Query (Server)

\[(answer, \text{proof}) \leftarrow \text{PerformQuery}(\mathcal{X}_j, \theta_j, \text{query})\]

- if \(\text{query} = x \in \mathcal{X}\):
  
  set \(\text{answer} = 1\) and \(\text{proof} = (\sigma_j)^{\frac{1}{s+x}} = g^{\frac{r \cdot \text{Ch}_{\mathcal{X}}(s)}{(s+x)}}. O(N \log N)\)

- if \(\text{query} = x \notin \mathcal{X}\):

  1. Using the Extended Euclidean algorithm, compute polynomials \(q_1[z], q_2[z]\) such that \(q_1[z] \text{Ch}_{\mathcal{X}}[z] + q_2[z](z + x) = 1. O(N \log^2 N \log \log N)\)

  2. Pick a random \(\gamma \leftarrow \mathbb{Z}_p^{*}\)

  3. Set \(q'_1[z] = q_1[z] + \gamma \cdot (z + x)\)

  4. Set \(q'_2[z] = q_2[z] - \gamma \cdot \text{Ch}_{\mathcal{X}}[z]\).

  5. Set \(W_1 := g^{q'_1(s)r^{-1}}, W_2 = g^{q'_2(s)}\).

  6. Set \(\text{proof} := (W_1, W_2)\) and \(\text{answer} = 0.\)
Verification (Client)

(accept/reject) ← Verify(pk, σ_j, query, answer, proof)

- Let query = x.
- If answer = 1, return accept if $e(σ_j, g) = e(\text{proof}, g^x \cdot pk)$. [O(1)]
- if answer = 0, return accept if $e(W_1, σ_j)e(W_2, g^x \cdot pk) = e(g, g)$. [O(1)]
- Return reject otherwise.
Update

$$(\mathcal{X}_{i+1}, \sigma_{i+1}, \text{upd}_i, \text{state}_{i+1}) \leftarrow \text{Update}(sk, \text{state}_i, \sigma_i, \theta_i, \mathcal{X}_i, u_i)$$

Owner:

- Choose $r' \leftarrow \mathbb{Z}_p^*$.
- If $x$ is to be inserted:
  1. Compute $\sigma_{i+1} = \sigma_i^{(s+x)r'}$. \(O(1)\)
- If $x$ is to be deleted:
  1. Compute $\sigma_{i+1} = \sigma_i^{r'}$. \(O(1)\)
- Set $\text{upd}_i = (r')$ and $\text{state}_{i+1} = \mathcal{X}_{i+1}$.

Server:

Store the inserted/deleted element and $\text{upd}_i = (r')$. \(O(1)\)
Privacy comes almost for free

<table>
<thead>
<tr>
<th></th>
<th>[Nguyen05 – No Privacy]</th>
<th>This work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>NMUL</td>
<td>NMUL</td>
</tr>
<tr>
<td>Update</td>
<td>1MUL</td>
<td>2MUL</td>
</tr>
<tr>
<td>Witness (Member)</td>
<td>NMUL + (N − 1)ADD</td>
<td>NMUL + (N − 1)ADD</td>
</tr>
<tr>
<td>Witness (Non-Member)</td>
<td>NMUL + (N − 1)ADD</td>
<td>(N + 1)MUL + (N − 1)ADD</td>
</tr>
<tr>
<td>Verify (Member)</td>
<td>1(MUL + ADD + PAIR)</td>
<td>1(MUL + ADD + PAIR)</td>
</tr>
<tr>
<td>Verify (Non-Member)</td>
<td>2(MUL + ADD + PAIR)</td>
<td>1(MUL + ADD + ADD₁) + 2PAIR</td>
</tr>
<tr>
<td>Witness Update (Member)</td>
<td>1(MUL + ADD)</td>
<td>2MUL + 1ADD</td>
</tr>
<tr>
<td>Witness Update (Non-Member)</td>
<td>2MUL + 1ADD</td>
<td>(N + 1)MUL + (N − 1)ADD</td>
</tr>
</tbody>
</table>

**Figure:** ADD = point addition MUL = scalar multiplication in the elliptic curve group G, ADD₁ = point addition in G₁ and PAIR a pairing computation, whereas N is the size of the set.
Set Algebra : Union
Query

\{X_1, \ldots, X_m\} = \text{set collection}

Client Query: Return union of sets 2, 5, 9

Server Response: \(\text{answer} = X_2 \cup X_5 \cup X_9\) + proof

Let \(X_2 = \{a, b, d\}, X_5 = \{d, f\}, X_9 = \{a, c\}\)

answer = \(\{a, c, b, d, f\}\)
Completeness Conditions

**Superset condition:** \( \mathcal{X}_2 \subseteq \text{answer} \land \mathcal{X}_5 \subseteq \text{answer} \land \mathcal{X}_9 \subseteq \text{answer} \).

**Technique:** Generalization of set membership.

**Membership condition:** \( \text{answer} \subseteq \tilde{U} \) where \( \tilde{U} = \mathcal{X}_2 \uplus \mathcal{X}_5 \uplus \mathcal{X}_9 \).
Proving membership

Multiset union: $\tilde{U} = \{a, a, c, c, b, d, d, f\}$

1. Compute
   \[ \sigma_{\tilde{U}} \leftarrow g(r_2 r_5 r_9) \text{Ch}_{\tilde{U}}(s) = g(r_2 r_5 r_9) (s+a)^2(s+c)(s+b)(s+d)^2(s+f) \]
Proving membership

Multiset union: $\tilde{U} = \{a, a, c, c, b, d, d, f\}$

1. Compute
   $$\sigma_{\tilde{U}} \leftarrow g(r_2r_5r_9)C_{\tilde{U}}(s) = g(r_2r_5r_9)(s+a)^2(s+c)(s+b)(s+d)^2(s+f)$$

2. Prove $\sigma_{\tilde{U}}$ is correctly computed
Proving membership

Multiset union: \( \tilde{U} = \{a, a, c, c, b, d, d, f\} \)

1. Compute
   \[
   \sigma_{\tilde{U}} \leftarrow g(r_2 r_5 r_9) \text{Ch}_{\tilde{U}}(s) = g(r_2 r_5 r_9)(s+a)^2(s+c)(s+b)(s+d)^2(s+f)
   \]

2. Prove \( \sigma_{\tilde{U}} \) is correctly computed

3. Prove \( \text{answer} \subseteq \tilde{U} \) using \( \sigma_{\tilde{U}} \)
Step 2: Server

\[ \sigma_2 = g^{r_2(s+a)(s+b)(s+d)} \]

\[ \sigma_5 = g^{r_5(s+d)(s+f)} \]

\[ \sigma_9 = g^{r_9(s+a)(s+c)} \]
Step 2: Server

\[ \sigma_{\bar{U}} = g^{(r_2 r_5 r_9)(s+a)^2(s+c)(s+b)(s+d)^2(s+f)} \]

\[ \sigma_{2,5} = g^{r_2 r_5 (s+a)(s+b)(s+d)^2(s+f)} \]

\[ \sigma_9 = g^{r_5 (s+a)(s+c)} \]

\[ \sigma_2 = g^{r_2 (s+a)(s+b)(s+d)} \]

\[ \sigma_5 = g^{r_5 (s+d)(s+f)} \]
Step 2: Client

\[ e(\sigma_{2,5}, \sigma_9) \equiv e(\sigma_{\tilde{u}}, g) \]

\[ e(\sigma_2, \sigma_5) \equiv e(\sigma_{2,5}, g) \]
Step 3

Server:  \( W_{(answer, \tilde{U})} \leftarrow g^{r_2 r_5 r_9 \text{Ch}_{\tilde{U}}(s)} \text{Ch}_{\text{answer}}(s) = g^{r_2 r_5 (s+a)(s+d)} \)

Client:  \( e(W_{(answer, \tilde{U})}, g^{\text{Ch}_{\text{answer}}(s)}) \equiv e(\sigma \tilde{U}, g) \)
More in the paper:

1. Relation of Zero Knowledge Accumulator with the existing primitives (ZKS, PSR, Trapdoorless Acc).
2. Formal proof that Zero knowledge is stronger than indistinguishably notion [MLPP12, DHS15] of privacy.
3. First efficient construction for zero-knowledge verifiable set algebra queries (Is-subset, Intersection, Union, Difference) with no additional cost over the state-of-the art non-private construction [PTT11].
Thank you!