Verifiable Order Queries on a List in Zero-Knowledge

Esha Ghosh

Brown University

Joint work with:
Olga Ohrimenko, Microsoft Research
Roberto Tamassia, Brown University

January 13, 2015
Overview
Motivation
Zero-Knowledge List (ZKL)
  ZKL Model and Security
  ZKL Construction
  ZKL Performance
Privacy-Preserving Authenticated List Model
Traditional Authenticated Data Structure
Efficiency Goals
PPAL-Algorithms
Tools for Construction
Construction
  High Level Idea
  Construction Details
Security Proofs
Summary
References
Table of Contents

Motivation

Zero-Knowledge List (ZKL)
  ZKL Model and Security
  ZKL Construction
  ZKL Performance

Privacy-Preserving Authenticated List Model

Traditional Authenticated Data Structure

Efficiency Goals

PPAL-Algorithms

Tools for Construction

Construction
  High Level Idea
  Construction Details

Security Proofs

Summary

References
Outsourced Data

Collaborate
Sales Reports

- Cloud
  - Revealing anything more?
  - Authentic?
  - Product I sold more than product II in March

- Company
- Client
- Sales / Marketing
# Health Records

<table>
<thead>
<tr>
<th>Vaccine</th>
<th>Date given (mm/dd/yy)</th>
<th>Administered by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hepatitis B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diptheria, Tetanus</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Haemophilus influenza type b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Polio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rotavirus</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Measles, Mumps and Rubella</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Varicella</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hepatitis A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meningococcal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Human papillomavirus</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Revealing anything more?

Authentic?
Proving values above a threshold
Other Statistical Queries - Max and Min

Let $S$ be a collection of elements of a totally ordered set, $\mathcal{L}$.

Max($S$) and Min($S$) return the highest ranked and the lowest ranked element of $S$ with respect to the list ordering.

The proof should reveal nothing beyond the answer, i.e., every element in $S$ is ranked lower than Max($S$).
Other Statistical Queries - top-\(t\)

The query \(\text{Top}(t, S)\) returns the top \(t\) ranked elements along with a proof of the answer.

The proof should not even reveal the order among the returned elements.
Model
Structured Data

Data is stored in *datastructures* like ordered sets, ordered trees.

```
<courses>
  <course ref='sp'>
  </course>
  <course ref='au'>
  </course>
</courses>
```

```
  <quarter id='sp'>
    <number>670</number>
    <days>TR</days>
  </quarter>
  <quarter id='au'>
    <number>680</number>
    <days>MWF</days>
  </quarter>
```

Spring

Autumn
Lists

A list is an ordered set of distinct elements.

\[ \mathcal{L} = \{y_1, y_2, \ldots, y_n\} \].

Example: Student id’s in a course, sorted in the non-decreasing order of their test score.
Table of Contents

Motivation

Zero-Knowledge List (ZKL)
  ZKL Model and Security
  ZKL Construction
  ZKL Performance

Privacy-Preserving Authenticated List Model

Traditional Authenticated Data Structure

Efficiency Goals

PPAL-Algorithms

Tools for Construction

Construction
  High Level Idea
  Construction Details

Security Proofs

Summary

References
Zero-Knowledge List (ZKL)

PHASE 1:
- Linearly Ordered List (L)
- “Commitment” made public
- Query (Member + Order) on L

PHASE 2:
- Answer + Proof
- Revealing anything more?
- Consistent with “commitment”?
Zero-Knowledge List (ZKL)

**PHASE 1:**
- Linearly Ordered List (L)
- "Commitment" made public

**PHASE 2:**
- Query (Member + Order) on L
- Answer + Proof

**PROVER**: (MALICIOUS)
- Owner
- Server

**VERIFIER**: (MALICIOUS)
- Client

Revealing anything more?
Consistent with "commitment"?
Query Operations

\( \mathcal{L} = \) linearly ordered list of elements:

**Membership Query**

*Client Query:* Is element \( y \in \mathcal{L} \)?

*Server Response:* If \( y \in \mathcal{L} \) return \( \mathcal{L}(y) = \text{true} + \text{proof of } y \in \mathcal{L} \); else return \( \mathcal{L}(y) = \text{false} + \text{proof of } y \notin \mathcal{L} \);

**Order Query**

*Client Query:* A pair of query elements \((y, w)\)

*Server Response:* If \( y, w \in \mathcal{L}, (y, w) \) rearranged according to their order in \( \mathcal{L} \) + membership proof + order proof. Proof of non-membership otherwise.
Security Properties

**Completeness:** Honestly generated proofs are always accepted by the client.

**Soundness:** Proofs for answers to queries, inconsistent with the initial commitment do not pass the verification.

**Zero-Knowledge:** Proofs do not reveal anything beyond the answers, i.e., the proofs are simulatable.
Zero-Knowledge Set

First let us consider an unordered finite set $S$ of key value pairs.

$S(x) = v$ denotes $v$ is the value corresponding to the key $x$. For the keys that are not present in $S$, $x \not\in S$, we write $S(x) = \bot$.

Queries are of the form “is key $x$ in $S$?”.

Return $S(x)$ if $x \in S$ + proof.

Return $\bot$ + proof if $x \not\in S$. 
Zero-Knowledge protocol to prove non-negativity of an integer

The prover sends a commitment $c$ to a non-negative value $x$ to the verifier and proves, without opening $c$, that $x \geq 0$.

We concisely write this as: $P \leftrightarrow V(x, r : c = C(x; r) \land x \geq 0)$.

The protocol is constructed on two facts:

- a negative number cannot be a sum of squares
- every non-negative integer is a sum of four squared integers
ZKL Setup

Let $\mathcal{L} = \{y_1, \ldots, y_n\}$ and $\text{rank}(\mathcal{L}, y_j) = j$.

For every $y_j \in \mathcal{L}$, compute $\mathbb{H}(y_j)$ and $C(j)$.

Instantiate ZKS with $(\mathbb{H}(y_j), C(j))$ as (key,value) pair.
Order Query in ZKL

Let $\mathcal{L} = \{Alice, Bob, Charles\}$.

Proving $Alice < Charles \in \mathcal{L}$:

As a part of the proof of membership membership of Alice, the verifier receives $C(1)$ and (3) for Bob.

These commitments are never opened.

Additionally, the prover returns a fresh commitment to 1, $C^*(1)$, and its opening information $\rho$.

Prover constructs proof using $P \leftrightarrow V(x, r : c = C(x; r) \land x \geq 0)$ to convince the verifier that $C(3 - 1 - 1)$ is a commitment to value $\geq 0$. 
Verification

The verifier verifies the commitment $C^*(1)$ using $\rho$.

The verifier computes $C(3 - 1 - 1) := C(3)/(C(1) \times C^*(1))$ using the homomorphic property of the integer commitment scheme.

Then it verifies the proof of non-negativity of $C(3 - 1 - 1)$ following the steps of the NIZK protocol $P \leftrightarrow V(x, r : c = C(x; r) \land x \geq 0)$. 

Performance

- The prover executes the commitment phase in time and space proportional to ZKS commitment.
- In the query phase, the prover computes the proof of the answer in time and space proportional to ZKS prover.
- The verifier verifies the proof in time and space proportional to ZKS verifier.
The construction has the following performance, where \( n \) is the list size, \( m \) is the query size, each element of the list is a \( k \)-bitstring and \( N \) is the number of all possible \( k \)-bit strings:

- The prover executes the commitment phase in \( O(n \log N) \) time and space.
- In the query phase, the prover computes the proof of the answer in \( O(m \log N) \) time.
- The verifier verifies the proof in \( O(m \log N) \) time and space.
# Table of Contents

- Motivation
- Zero-Knowledge List (ZKL)
  - ZKL Model and Security
  - ZKL Construction
  - ZKL Performance
- Privacy-Preserving Authenticated List Model
- Traditional Authenticated Data Structure
- Efficiency Goals
- PPAL-Algorithms
- Tools for Construction
- Construction
  - High Level Idea
  - Construction Details
- Security Proofs
- Summary
- References
Model

**OWNER** (TRUSTED)  **SERVER** (MALICIOUS)  **CLIENT** (MALICIOUS)

Linearly Ordered List (L)

1 2 3 4 5 6 7

Server Digest

Client Digest

Order Query on L

Answer + Proof

Revealing anything more?

Authentic?
Query

\[ \mathcal{L} = \text{linearly ordered list of elements:} \]

**Order Query**

*Client Query:* A pair of query elements \((x, y)\) of \(\mathcal{L}\)

*Server Response:* \((x, y)\) rearranged according to their order in \(\mathcal{L}\) + order proof
Adversarial Model

Both the server and the client can act as adversaries:

*Server*: May try to forge proofs for incorrect answers to membership or ordering queries

*Client*: Tries to learn from the proofs additional information about list $\mathcal{L}$ beyond what she has asked for
Security Properties

Completeness: Honestly generated proofs are always accepted by the client.

Soundness: Proofs for incorrect answers forged by the server do not pass the verification by the client.

Zero-Knowledge: Proofs do not reveal anything beyond the answers, i.e., the proofs are simulatable.
Table of Contents

Motivation

Zero-Knowledge List (ZKL)
- ZKL Model and Security
- ZKL Construction
- ZKL Performance

Privacy-Preserving Authenticated List Model

Traditional Authenticated Data Structure

Efficiency Goals

PPAL-Algorithms

Tools for Construction

Construction
- High Level Idea
- Construction Details

Security Proofs

Summary

References
Merkle’s Hash Tree

Merkle’s Hash Tree where the elements are stored at the leaves in sorted order reveals the size of the source list and the rank of an element.
Authenticated skip list

The proof of membership/non-membership is a sequence of nodes visited when searching for the element.

Figure: Elements searched for the value 86 are shown in red.
Table of Contents

Motivation
Zero-Knowledge List (ZKL)
  ZKL Model and Security
  ZKL Construction
  ZKL Performance
Privacy-Preserving Authenticated List Model
Traditional Authenticated Data Structure

Efficiency Goals

PPAL-Algorithms
Tools for Construction
Construction
  High Level Idea
  Construction Details

Security Proofs
Summary
References
Efficiency Goals

Let \( n \) = number of data elements in the source list \( \mathcal{L} \) and let \( m = \) number of elements in the queried sublist

- **Server storage**: \( O(n) \), irrespective of the number of queries answered
- **Proof size**: proportional to \( O(m) \)
- **Setup time**: proportional to \( O(n) \)
- **Query time**: proportional to \( O(m) \)
- **Verification time**: proportional to \( O(m) \)
Related Work

<table>
<thead>
<tr>
<th></th>
<th>[SBZ01]</th>
<th>[JMSW02]</th>
<th>[CLX09]</th>
<th>[BBD+10]</th>
<th>[SPB+12]</th>
<th>[PSPDM12]</th>
<th>[KAB12]</th>
<th>This Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obliviousness</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Setup time</td>
<td>$n \log n$</td>
<td>$n$</td>
<td>$n$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n$</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>Space</td>
<td>$n$</td>
<td>$n$</td>
<td>$n$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n$</td>
<td>$n^2$</td>
<td>$n$</td>
</tr>
<tr>
<td>Query time</td>
<td>$m$</td>
<td>$n \log n$</td>
<td>$n$</td>
<td>$mn$</td>
<td>$m$</td>
<td>$n$</td>
<td>$n$</td>
<td>$\min(m, \log n)$</td>
</tr>
<tr>
<td>Verification time</td>
<td>$m \log n \log m$</td>
<td>$m \log n$</td>
<td>$n^2$</td>
<td>$m^2$</td>
<td>$m^2$</td>
<td>$m$</td>
<td>$m$</td>
<td>$m$</td>
</tr>
<tr>
<td>Integrity proof</td>
<td>$m$</td>
<td>$m \log n$</td>
<td>$n$</td>
<td>$m^2$</td>
<td>$m$</td>
<td>$m$</td>
<td>$m$</td>
<td>$m$</td>
</tr>
<tr>
<td>Order proof</td>
<td>$m$</td>
<td>$m \log n$</td>
<td>$n$</td>
<td>$m^2$</td>
<td>$m^2$</td>
<td>$m$</td>
<td>$n$</td>
<td>$m$</td>
</tr>
<tr>
<td>Assumption</td>
<td>RSA</td>
<td>RSA</td>
<td>SRSA Division</td>
<td>EUCMA</td>
<td>nEAE</td>
<td>AnAHF</td>
<td>ROH, RSA</td>
<td>nBDHI</td>
</tr>
</tbody>
</table>

Table: Comparison with previous works where $n = \text{size of the list}$, $m = \text{size of the queried sublist}$
Table of Contents

Motivation
Zero-Knowledge List (ZKL)
  ZKL Model and Security
  ZKL Construction
  ZKL Performance
Privacy-Preserving Authenticated List Model
Traditional Authenticated Data Structure
Efficiency Goals
PPAL-Algorithms
Tools for Construction
Construction
  High Level Idea
  Construction Details
Security Proofs
Summary
References
Privacy Preserving Authenticated List (PPAL)

\[ \mathcal{L} = \text{ordered list of distinct elements.} \]

- \((\text{digest}_C, \text{digest}_S) \leftarrow \text{Setup}(1^k, \mathcal{L})(\text{Owner})\)
  \[ \text{digest}_S = \text{Digest for the server} \]
  \[ \text{digest}_C = \text{Digest for the client} \]

- \((\text{order}, \text{proof}) \leftarrow \text{Query}(\text{digest}_S, \mathcal{L}, \delta)(\text{Server})\)
  \[ \delta = \text{sublist of a list } \mathcal{L} \text{ is defined as: } \text{Elements}(\delta) \subseteq \text{Elements}(\mathcal{L}). \]
  \[ \text{order} = \pi_{\mathcal{L}}(\delta) \]
  \[ \text{proof} = \text{proof of order.} \]
Privacy Preserving Authenticated List (PPAL) (Cont.)

- $b \leftarrow \text{Verify}(\text{digest}_C, \delta, \text{order}, \text{proof})$ (Client)
  
  $b = \text{ACCEPT}$ iff $\text{Elements}(\delta) \subseteq \text{Elements}(L)$ and $\text{order} = \pi_L(\delta)$.
  Otherwise, $b = \text{REJECT}$. 
Table of Contents

Motivation
Zero-Knowledge List (ZKL)
  ZKL Model and Security
  ZKL Construction
  ZKL Performance
Privacy-Preserving Authenticated List Model
Traditional Authenticated Data Structure
Efficiency Goals
PPAL-Algorithms

Tools for Construction

Construction
  High Level Idea
  Construction Details
Security Proofs
Summary
References
Aggregate Signature

Given signatures $\sigma_1, \ldots, \sigma_n$ on distinct messages $M_1, \ldots, M_n$ from $n$ distinct users $u_1, \ldots, u_n$, an aggregate signature aggregates these signatures into a single short signature $\sigma$.

$\sigma$ and the $n$ messages convince the verifier that the $n$ users indeed signed the $n$ original messages (i.e., user $i$ signed message $M_i$).

We use the special case where a single user signs $n$ distinct messages $M_1, \ldots, M_n$.

Security: the aggregate signature $\sigma$ is valid if and only if the aggregator used all $\sigma_i$'s to construct it.
Table of Contents
Motivation
Zero-Knowledge List (ZKL)
  ZKL Model and Security
  ZKL Construction
  ZKL Performance
Privacy-Preserving Authenticated List Model
Traditional Authenticated Data Structure
Efficiency Goals
PPAL-Algorithms
Tools for Construction
Construction
  High Level Idea
  Construction Details
Security Proofs
Summary
References
Intuition

Every element of the list is associated with a member witness.

The rank of the element in encoded the member witness and then the rank information \textit{“blinded” with randomness}.

Every pair of \textit{(element, member witness)} is signed by the owner.

The signatures are aggregated to compute the list digest signature.
Intuition (Continued)

The list digest signature and some information is sent to the server.

The list digest signature is sent to the client.

The server can compute a valid digest signature for any sublist of the source list by exploiting the homomorphic nature of aggregate signatures, that is without owner’s involvement.

The client can verify the individual signatures in a single invocation to aggregate signature verification.
Intuition (Continued)

The owner sends to the server the random elements used in computing the member witnesses.

The server computes the order witnesses without the owner’s involvement, using these random elements.
Schematically

Unique list identifier = $\omega$

Member witness

$t_1 \sim (1, x_1) \quad t_2 \sim (2, x_2) \quad t_3 \sim (3, x_3) \quad t_4 \sim (4, x_4)$

Owner signs each member witness

Sig

Order witness

$t(x_1 < x_3) \sim (t_1, t_3)$

Computed by server

List digest signature computed by owner

Sig

Sig
Some Notations

- $k \in \mathbb{N}$ 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
- $\mathcal{H} : \{0, 1\}^* \rightarrow G$: full domain hash function
- System parameters: $(p, G, G_1, e, g, \mathcal{H})$
- We assume all the efficiently computable group operations and hashing can be performed in $O(1)$ time (abstracting the $O(poly(k))$ time and $O(poly(k))$ space required for representation of group elements)
Setup

\[(\text{digest}_C, \text{digest}_S) \leftarrow \text{Setup}(1^k, \mathcal{L})\]

**Input:** The security parameter $1^k$ and the list, $\mathcal{L} = \{x_1, \ldots, x_n\}$

**Algorithm:**

- The secret key $\text{sk}_o = \langle s, v \rangle$, $s \leftarrow \mathbb{Z}_p^*$ and $v \leftarrow \mathbb{Z}_p^*$ \[O(1)\]
- $g \leftarrow G \[O(1)\]$ For every element $x_i$ in $\mathcal{L} = \{x_1, \ldots, x_n\}$
  - $r_i \leftarrow \mathbb{Z}_p^* \forall i : O(n)$
  - Set $\Omega_\mathcal{L} := \langle r_1, r_2, \ldots, r_n \rangle$, $r_i \neq r_j$ for $i \neq j$
  - Member witness, $t_{x_i \in \mathcal{L}} \leftarrow (g^{s_i})_{r_i} \forall i : O(n)$
Setup (Continued)

Algorithm:

- Sign $x_i$ as $\sigma_i \leftarrow H(t_{x_i \in \mathcal{L}} \| x_i)^v \forall i : O(n)$
- Pick nonce $\omega \leftarrow \{0, 1\}^*$ and salt $\leftarrow H(\omega)^v \ O(1)$
- List digest signature: $\sigma_\mathcal{L} \leftarrow \text{salt} \times \prod_{1 \leq i \leq n} \sigma_i \ O(n)$
- Set $\Sigma_\mathcal{L} := \langle \{ t_{x_i \in \mathcal{L}}, \sigma_i \}_{1 \leq i \leq n}, H(\omega) \rangle$
- Compute $\langle g, g^s, g^{s^2}, \ldots, g^{s^n} \rangle \ O(n)$
- Set $\text{digest}_C := (g^v, \sigma_\mathcal{L})$
- Set $\text{digest}_S := (g^v, \sigma_\mathcal{L}, \langle g, g^s, g^{s^2}, \ldots, g^{s^n} \rangle, \Sigma_\mathcal{L}, \Omega_\mathcal{L})$
- Return $(\text{digest}_C, \text{digest}_S)$
Query

Unique list identifier = \( \omega \)

Member witness

\( t_1 \sim (1, x_1) \)
\( t_2 \sim (2, x_2) \)
\( t_3 \sim (3, x_3) \)
\( t_4 \sim (4, x_4) \)

Server

\( \Sigma_1 \)
\( \Sigma_3 \)
\( \Sigma_d \)

Member verification unit

\( L' = L \setminus d \)

Queried sublist = \( d \)
Query (Continued)

(order, proof) ← Query(digest_S, L, δ)

Input:

- The server digest, digest_S
- The list L
- A sublist, \( \delta = \{z_1, \ldots, z_m\}, z_i \in L, 1 \leq i \leq m \).

Algorithm:

- Let \( L' = L \setminus \delta \)
- \( \text{order} \leftarrow \pi_L(\delta) = \{y_1, \ldots, y_m\} \quad O(m) \)
- \( \Sigma_{\text{order}} = \langle \sigma_{\text{order}}, T, \lambda_{L'} \rangle \)
- \( T = \{t_{y_1 \in L}, \ldots, t_{y_m \in L}\} \)
- Digest signature for the sublist \( \sigma_{\text{order}} \leftarrow \prod_{z_j \in \delta} \sigma_{\text{rank}(L, z_j)} \quad O(m) \)
- Member verification unit: \( \lambda_{L'} \leftarrow H(\omega) \times \prod_{x \in L'} H(t_{x_{\text{rank}(L, x)} \in L} || x) \quad O(n - m), \text{ With preprocessing: } O(\min\{m \log n, n\}) \)
- Set \( \Sigma_{\text{order}} := (\sigma_{\text{order}}, T, \lambda_{L'}) \)
Query (Continued)

Unique list identifier = \( \omega \)

Member witness:

- \( t_1 \sim (1, x_1) \)
- \( t_2 \sim (2, x_2) \)
- \( t_3 \sim (3, x_3) \)
- \( t_4 \sim (4, x_4) \)

Order witness:

- \( t_{(x_1 < x_3)} \sim (|3-1|, t_1, t_3) \)

Queried sublist = \( d' \)

\( \mathcal{L}' = \mathcal{L} \setminus d' \)

Distance between \( X_1 \) and \( X_3 \)
Query (Continued)

Algorithm:

- Let $i' = \text{rank}(\mathcal{L}, y_j), i'' = \text{rank}(\mathcal{L}, y_{j+1})$
- Let $d = |i' - i''|, r_1 = \Omega_{\mathcal{L}}[i']^{-1}$ and $r_2 = \Omega_{\mathcal{L}}[i'']$
- Order witness, $t_{y_j < y_{j+1}} = (g^{s^d})^r_1 r_2 \forall i : O(m)$
- Set $\Omega_{\text{order}} := \{t_{y_1 < y_2}, t_{y_2 < y_3}, \ldots, t_{y_{m-1} < y_m}\}$.
- Set proof := $(\Sigma_{\text{order}}, \Omega_{\text{order}})$
- Return (order, proof)
Verify

Unique list identifier = $\omega$

Member witness

Queried sublist = $d$

$L' = L \setminus d$

Client

(1) Verify $\text{Sig}_d$

(2) Verify if $\text{Combined} = ?$

Member verification unit

$\text{Sig}_d$
Verify (Continued)

Let $b \leftarrow \text{Verify}(\text{digest}_C, \delta, \text{order}, \text{proof})$

**Input:**
- The client digest $\text{digest}_C$
- A sublist $\delta = \{y_1, \ldots, y_m\}$
- A permutation on the elements of $\delta$, $\text{order}$
- Proof of the order, $\text{proof}$

**Algorithm:**
- Check if $\text{Elements}(\text{order}) \equiv \text{Elements}(\delta)$ \(O(m)\)
- Compute $\xi \leftarrow \prod_{y_j \in \delta} \mathcal{H}(t_{y_j \in \mathcal{L}} \| y_j)$ \(O(m)\)
- Check if $e(\sigma_{\text{order}}, g) \equiv e(\xi, g^\nu)$ and $e(\sigma_{\mathcal{L}}, g) \equiv e(\sigma_{\text{order}}, g) \times e(\lambda_{\mathcal{L}'}, g^\nu)$ \(O(1)\)
Verify (Continued)

Unique list identifier = \( \omega \)

Member witness:

- \( t_1 \sim (1, x_1) \)
- \( t_2 \sim (2, x_2) \)
- \( t_3 \sim (3, x_3) \)
- \( t_4 \sim (4, x_4) \)

Order witness:

- \( t(x_1 < x_3) \sim (|3-1|, t_1, t_3) \)

Client:

- \( t_1 \sim (1, x_1) \)
- \( t(x_1 < x_3) \sim (|3-1|, t_1, t_3) \)
- \( t_3 \sim (3, x_3) \)

Verify if \( \text{Rank}(X_1) + \text{Distance between } X_1 \text{ and } X_3 =? \text{Rank}(X_3) \)
Verify (Continued)

**Algorithm:**

- For every $j \in [1, m - 1]$, $e(t \in L, t \in \leq y_{j+1}) = e(t \in L, t \in g)$. $O(m)$
- Return ACCEPT iff all the equalities verify, and REJECT, otherwise.
The server can precompute and store some products to reduce the overall running time of this algorithm to $O(m \log n)$ when $m \ll n$.

Let $\psi_i = H(t_{x_i \in L}||x_i)$ for every element in $L = \{x_1, \ldots, x_n\}$.

A balanced binary tree is built over $n$ leaves, where the $ith$ leaf and stores $\psi_i$.

Each internal node of the tree stores the product of the values stored at its children.
Preprocessing Step (Continued)
Preprocessing Step (Continued)

Computing each internal node takes time $O(1)$ and the tree has $O(n)$ nodes.

So, the precomputation takes time $O(n)$ and requires $O(n)$ storage.

Computing $\lambda_{L'}$ will require computing the product of $m + 1$ partial products.

Each partial product can be computed using $O(\log n)$ precomputed products.

The total time required to compute the product of $m + 1$ partial products is $O((m + 1) \log n) = O(m \log n)$. 
Complexity Summary

*Table:* $n =$ number of elements in the source list $\mathcal{L}$ and $m \leq n$, size of the queried sublist

<table>
<thead>
<tr>
<th></th>
<th>Time Complexity</th>
<th>Space Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Server</td>
<td>Preprocessing: $O(n)$, $O(\min{m \log n, n})$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Client</td>
<td>$O(m)$</td>
<td>$O(m)$</td>
</tr>
</tbody>
</table>
Table of Contents

Motivation
Zero-Knowledge List (ZKL)
  ZKL Model and Security
  ZKL Construction
  ZKL Performance
Privacy-Preserving Authenticated List Model
Traditional Authenticated Data Structure
Efficiency Goals
PPAL-Algorithms
Tools for Construction
Construction
  High Level Idea
  Construction Details
Security Proofs
Summary
References
Hardness Assumption

Definition ($n$-Bilinear Diffie Hellman Inversion ($n$-BDHI) [BB04])

Let $s$ be a random element of $\mathbb{Z}_p^*$ and $n$ be a positive integer. Then, for every PPT adversary $A$ there exists a negligible function $\nu(.)$ such that:

$$\Pr[s \leftarrow \mathbb{Z}_p^*; y \leftarrow A(\langle g, g^s, g^{s^2}, \ldots, g^{s^n} \rangle) : y = e(g, g)^{\frac{1}{s}}] \leq \nu(k).$$

Dan Boneh and Xavier Boyen.
Efficient selective-id secure identity based encryption without random oracles.
Soundness Proof Sketch

We assume that there exists a malicious server adv, which forges the order on a non-trivial sublist $\delta = \{x_1, \ldots, x_m\}$, where $m \geq 2$, for a list $\mathcal{L}$.

Then there exists at least one inversion pair $(x_i, x_j)$ whose order is flipped in adv’s forgery.

Wlog assume that $u < v$ where $u = \text{rank}(\mathcal{L}, x_i)$ and $v = \text{rank}(\mathcal{L}, x_j)$. 
Soundness Proof Sketch (Continued)

Then adv must have forged the witness $t_{x_j < x_i} = (g^{s(u-v)})^{r_1 r_2^{-1}}$ that passes the verification, where $r_1, r_2 \in \mathbb{Z}_p^*$ are the randomnesses used in member witnesses of elements $x_i$ and $x_j$, respectively.

By invoking adv and using its forged witness $t_{x_j < x_i}$, we can construct a PPT adversary that successfully breaks the $n$-BDHI hardness assumption by outputting

$$e(t_{x_j < x_i}, (g^{s(v-u-1)})^{r_1^{-1} r_2}) = e(g, g)^{\frac{1}{s}}$$
Zero-Knowledge Proof Sketch

We write a stateful simulator Sim that has access to the system parameters \((p, G, G_1, e, g, \mathcal{H})\).

Sim picks a random element \(v \xleftarrow{\$} \mathbb{Z}_p^*\) and a random element \(g_1 \xleftarrow{\$} G\) and publishes as digest \(C = (g^v, g_1^v)\) and keeps \(v\) as the secret key.

For a query on sublist \(\delta = \{x_1, x_2, \ldots, x_m\}\), Sim makes an oracle access to list \(\mathcal{L}\) to get the list order of the elements.

Sim fakes the proof objects as follows:
Zero-Knowledge Proof Sketch (Continued)

- Sim picks a random element $r_i \leftarrow \mathbb{Z}_p^*$ sets the member authentication unit as $t_{y_i \in L} := g^{r_i}$.
- Sim computes $\sigma_{y_i} \leftarrow H(t_{y_i \in L} || y_i)^v$.
- For every pair of elements $y_i, y_{i+1}$ in order, Sim computes $t_{y_i < y_{i+1}} \leftarrow g^{r_{i+1}/r_i}$.

Thus Sim produces view identically distributed to real view of the adversary, given only oracle access to the list, using the fact that our PPAL construction uses witnesses blinded in their exponents.
Summary

Theorem

The privacy-preserving authenticated list (PPAL) construction of satisfies the security properties of completeness, soundness and zero-knowledge in the random oracle model and under the $n$-BDHI assumption. Also, the construction has the following performance, where $n$ denotes the list size and $m$ denotes the query size.

- The owner and the server use $O(n)$ space.
- The owner performs the setup phase in $O(n)$ time.
- The server performs the preprocessing phase in $O(n)$ time.
- The server computes the answer to a query and its proof in $O(\min\{m \log n, n\})$ time.
- The client verifies the proof in $O(m)$ time and space.
# Table of Contents

- Motivation
- Zero-Knowledge List (ZKL)
  - ZKL Model and Security
  - ZKL Construction
  - ZKL Performance
- Privacy-Preserving Authenticated List Model
- Traditional Authenticated Data Structure
- Efficiency Goals
- PPAL-Algorithms
- Tools for Construction
- Construction
  - High Level Idea
  - Construction Details
- Security Proofs
- Summary
- References
References

Ron Steinfeld, Laurence Bull, and Yuliang Zheng.
Content extraction signatures.

Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner.
Homomorphic signature schemes.

Ee-Chien Chang, Chee Liang Lim, and Jia Xu.
Short redactable signatures using random trees.

Christina Brzuska, Heike Busch, Ozgur Dagdelen, Marc Fischlin, Martin Franz, Stefan Katzenbeisser, Mark Manulis, Cristina Onete, Andreas Peter, Bertram Poettering, and Dominique Schröder.
Redactable signatures for tree-structured data: Definitions and constructions.

Kai Samelin, Henrich C. Poehls, Arne Bilzhauser, Joachim Posegga, and Hermann De Meer.
Redactable signatures for independent removal of structure and content.

Henrich C. Poehls, Kai Samelin, Joachim Posegga, and Hermann De Meer.
Length-hiding redactable signatures from one-way accumulators in $O(n)$.
Technical Report MIP-1201, Faculty of Computer Science and Mathematics (FIM), University of Passau, 2012.

Ashish Kundu, Mikhail J. Atallah, and Elisa Bertino.
Leakage-free redactable signatures.

Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham.
Aggregate and verifiably encrypted signatures from bilinear maps.

Dan Boneh and Xavier Boyen.
Efficient selective-id secure identity based encryption without random oracles.

Dan Boneh, Xavier Boyen, and Eu-Jin Goh.
Hierarchical identity based encryption with constant size ciphertext.
Thank you!