Strong Accumulators from Collision-Resistant Hashing

ISC 2008
Taipei - Taiwan

Philippe Camacho (University of Chile)
Alejandro Hevia (University of Chile)
Marcos Kiwi (University of Chile)
Roberto Opazo (CEO Acepta.com)
Outline

- Notion of accumulator
- Motivation
  - e-Invoice Factoring
- Our construction
- Conclusion
Notion of accumulator

Problem
- A set $X$.
- Given an element $x$ we wish to prove that this element belongs or not to $X$.

Let $X = \{x_1, x_2, \ldots, x_n\}$:
- $X$ will be represented by a short value $\text{Acc}$.
- $\text{Belongs} (\text{Acc}, x, w) = \text{True} \iff x$ belongs to $X$.

Witness
Notion of accumulator

- Accumulator Manager
  - Computes setup values.
  - Computes the accumulated value $\text{Acc}$.
  - Computes the witness $w_x$ for a given $x$.

- Accumulator Users
  - Check that an element belongs or not to the set, using $\text{Acc}$, $w_x$, and $x$. 
Applications

- Time-stamping [BeMa94]
- Certificate Revocation List [LLX07]
- Anonymous credentials [CamLys02]
- E-Cash [AWSM07]
- Broadcast Encryption [GeRa04]

...
Factoring Industry in Chile

Provider (Milk seller)

Factoring Entity

Client (Supermarket)

Nothing to see with Number Theory!
Factoring Industry in Chile

Factoring Entity

Provider
(Milk seller)

Client
(Supermarket)

1) I want (a lot of) milk now *

(*) but I do not want to pay yet.

Nothing to see with Number Theory!
Factoring Industry in Chile

Provider (Milk seller)

1) I want (a lot of) milk now *.

2) Here is your milk.

Client (Supermarket)

Factoring Entity

(*) but I do not want to pay yet.

Nothing to see with Number Theory!
Factoring Industry in Chile

Provider (Milk seller) → Factoring Entity

Client (Supermarket) ← 1) I want (a lot of) milk now (*).
2) Here is your milk.
3) Please pay the invoice.

(*) but I do not want to pay yet.
Factoring Industry in Chile

1) I want (a lot of) milk now (*).

2) Here is your milk.

3) Please pay the invoice.

4) Here is your money (**).

(*) but I do not want to pay yet.

(**) minus a fee.
Factoring Industry in Chile

Factoring Entity

Provider (Milk seller)

Client (Supermarket)

1) I want (a lot of) milk now (*).
2) Here is your milk.
3) Please pay the invoice.
4) Here is your money (**).
5) It’s time to pay.

(*) but I do not want to pay yet.
(**) minus a fee.

Nothing to see with Number Theory!
Factoring Industry in Chile

1) I want (a lot of) milk now (*).
2) Here is your milk.
3) Please pay the invoice.
4) Here is your money (**).
5) It’s time to pay.
6) Here is the money.

Provider
(Milk seller)

Factoring Entity

Client
(Supermarket)

(*) but I do not want to pay yet.
(**) minus a fee.

Nothing to see with Number Theory!
The Problem

- A malicious provider could send the same invoice to various Factoring Entities.

- Then he leaves to a far away country with all the money.

- Later, several Factoring Entities will try to charge the invoice to the same client. Losts must be shared…
Solution with Factoring Authority

Factoring Authority

(1) Invoice

(2) Ack

(3) Invoice, Ack

(4) Is there the invoice?

(5) YES / NO

FE_1

FE_2

... 

FE_i

... 

FE_n

Provider

Client
Caveat

- This solution is quite simple.

- However
  - Trusted Factoring Authority is needed.

- Can we remove this requirement?
Properties

- **Dynamic**
  - Allows insertion/deletion of elements.

- **Universal**
  - Allows proofs of membership and nonmembership.

- **Strong**
  - No need to trust in the Accumulator Manager.
### Prior work

<table>
<thead>
<tr>
<th></th>
<th>Dynamic</th>
<th>Strong</th>
<th>Universal</th>
<th>Security</th>
<th>Efficiency (witness size)</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BeMa94]</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>RSA + RO</td>
<td>O(1)</td>
<td>First definition</td>
</tr>
<tr>
<td>[BarPfi97]</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>Strong RSA</td>
<td>O(1)</td>
<td>-</td>
</tr>
<tr>
<td>[CamLys02]</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>Strong RSA</td>
<td>O(1)</td>
<td>First dynamic accumulator</td>
</tr>
<tr>
<td>[LLX07]</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>Strong RSA</td>
<td>O(1)</td>
<td>First universal accumulator</td>
</tr>
<tr>
<td>[AWSM07]</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>Pairings</td>
<td>O(1)</td>
<td>E-cash</td>
</tr>
<tr>
<td>[WWP08]</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>eStrong RSA Paillier</td>
<td>O(1)</td>
<td>Batch Update</td>
</tr>
</tbody>
</table>
## Prior work

<table>
<thead>
<tr>
<th></th>
<th>Dynamic</th>
<th>Strong</th>
<th>Universal</th>
<th>Security</th>
<th>Efficiency (witness size)</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BeMa94]</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>RSA + RO</td>
<td>O(1)</td>
<td>First definition</td>
</tr>
<tr>
<td>[BarPfi97]</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>Strong RSA</td>
<td>O(1)</td>
<td></td>
</tr>
<tr>
<td>[CamLys02]</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>Strong RSA</td>
<td>O(1)</td>
<td>First dynamic accumulator</td>
</tr>
<tr>
<td>[LLX07]</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>Strong RSA</td>
<td>O(1)</td>
<td>First universal accumulator</td>
</tr>
<tr>
<td>[AWSM07]</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>Pairings</td>
<td>O(1)</td>
<td>E-cash</td>
</tr>
<tr>
<td>[WWP08]</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>eStrong RSA Paillier</td>
<td>O(1)</td>
<td>Batch Update</td>
</tr>
<tr>
<td>[CHKO08]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Collision-Resistant Hashing</td>
<td>O(ln(n))</td>
<td>Our work</td>
</tr>
</tbody>
</table>
Notation

- $H: \{0,1\}^* \rightarrow \{0,1\}^k$
  - randomly chosen function from a family of collision-resistant hash functions.

- $x_1, x_2, x_3, \ldots \in \{0,1\}^k$
  - $x_1 < x_2 < x_3 < \ldots$ where $<$ is the lexicographic order on binary strings.

- $-\infty, \infty$
  - Special values such that
    - For all $x \in \{0,1\}^k$: $-\infty < x < \infty$

- $||$ denotes the concatenation operator.
Ideas

- **Merkle-trees**

  Root value:
  Represents the set \( \{x_1, \ldots, x_8\} \)

  \[
  Z_1 = H(Y_1 || Y_2) \\
  Z_2 = H(Y_3 || Y_4) \\
  P = H(Z_1 || Z_2)
  \]

  
  \[
  X_4 \rightarrow Y_1 = H(x_4 || x_1) \\
  X_5 \rightarrow Y_2 = H(x_5 || x_6) \\
  X_2 \rightarrow Y_3 = H(x_2 || x_8) \\
  X_7 \rightarrow Y_4 = H(x_7 || x_3)
  \]
Ideas

- Merkle-trees

Root value: Represents the set \{x_1, \ldots, x_8\}

\[ Z_1 = H(Y_1 || Y_2) \]
\[ Y_1 = H(x_4 || x_1) \]
\[ X_4 \quad X_1 \]
\[ Y_2 = H(x_5 || x_6) \]
\[ X_5 \quad X_6 \]
\[ Z_2 = H(Y_3 || Y_4) \]
\[ Y_3 = H(x_2 || x_8) \]
\[ X_2 \quad X_8 \]
\[ Y_4 = H(x_7 || x_3) \]
\[ X_7 \quad X_3 \]

\[ P = H(Z_1 || Z_2) \]

\[ O(\ln(n)) \]
Ideas

How to prove non-membership?

- Kocher’s trick [Koch98]: store pair of consecutive values

  - $X = \{1, 3, 5, 6, 11\}$
  - $X' = \{(-\infty, 1), (1, 3), (3, 5), (5, 6), (6, 11), (11, \infty)\}$
  - $y = 3$ belongs to $X \iff (1, 3)$ or $(-\infty, 1)$ belongs to $X'$.
  - $y = 2$ does not belong to $X \iff (1, 3)$ belongs to $X'$. 
Public Data Structure

- Called “Memory”.
- Compute efficiently the accumulated value and the witnesses.
- In our construction the Memory will be a binary tree.
How to insert elements?

\((-\infty, \infty)\)

\(X=\emptyset, \text{ next: } x_1\)
How to insert elements?

$X = \{x_1\}$, next: $x_2$
How to insert elements?

\[
X = \{x_1, x_2\}, \text{ next: } x_5
\]
How to insert elements?

$X = \{ x_1, x_2, x_5 \}$, next: $x_3$
How to insert elements?

$X = \{x_1, x_2, x_3, x_5\}$, next: $x_4$
How to insert elements?

$X = \{ x_1, x_2, x_3, x_4, x_5 \}$, next: $x_6$
How to insert elements?

\[ X = \{ x_1, x_2, x_3, x_4, x_5, x_6 \} \]
How to compute the accumulated value?

Proof_N = H(Proof_left || Proof_right || value)

Proof_Nil = ""

Acc = Proof.Root

A pair (x_i, x_j)
How to update the accumulated value? (Insertion)

Next element to be inserted: $x_8$

We will need to recompute proof node values.
How to update the accumulated value? (Insertion)

New element: \( x_8 \).

Proof\(_N\) stored in each node.

Dark nodes do not require recomputing Proof\(_N\).

Only a logarithmic number of values needs recomputation.
Security

- Consistency
  - Difficult to find witnesses that allow to prove inconsistent statements.
    - $X=\{1,2\}$
    - Hard to compute a membership witness for 3.
    - Hard to compute a nonmembership witness for 2.

- Update
  - Guarantees that the accumulated value represents the set after insertion/deletion of $x$. 
Security

- **Lemma**: Given a tree $T$ with accumulated value $\text{Acc}_T$, finding a tree $T', T \neq T'$ such that $\text{Acc}_T = \text{Acc}_{T'}$ is difficult.

- **Proof (Sketch)**: $\text{Proof}_N = H(\text{Proof}_{\text{left}} || \text{Proof}_{\text{right}} || \text{value})$
Security

- **Lemma**: Given a tree $T$ with accumulated value $\text{Acc}_T$, finding a tree $T', T \neq T'$ such that $\text{Acc}_T = \text{Acc}_{T'}$ is difficult.

- **Proof (Sketch)**: $\text{Proof}_N = H(\text{Proof}_{\text{left}} || \text{Proof}_{\text{right}} || \text{value})$

![Diagram showing a graph with nodes labeled with tuples and arrows indicating a collision for $H$.]
Security

- **Lemma**: Given a tree \( T \) with accumulated value \( \text{Acc}_T \), finding a tree \( T', T \neq T' \) such that \( \text{Acc}_T = \text{Acc}_{T'} \) is difficult.

- **Proof (Sketch)**: \( \text{Proof}_N = H(\text{Proof}_{\text{left}} \| \text{Proof}_{\text{right}} \| \text{value}) \)
Security

- **Lemma**: Given a tree $T$ with accumulated value $\text{Acc}_T$, finding a tree $T', T \neq T'$ such that $\text{Acc}_T = \text{Acc}_{T'}$ is difficult.

- **Proof (Sketch)**: $\text{Proof}_N = H(\text{Proof}_{\text{left}} \| \text{Proof}_{\text{right}} \| \text{value})$
Security (Consistency)

Witness: blue nodes and the \((x_3, x_4)\) pair, size in \(O(\ln(n))\)

Checking that \(x\) belongs (or not) to \(X\):

1) compute recursively the proof \(P\) and verify that \(P=\text{Acc}\)

2) check that: \(x=x_3\) or \(x=x_4\) (membership)

\(x_3 < x < x_4\) (nonmembership)
Security (Update)

Before

After

Insertion of $x_8$
Conclusion & Open Problem

- First *dynamic, universal, strong* accumulator.
- Simple.
- Security
  - Existence of collision-resistant hash functions.
- Solves the e-Invoice Factoring Problem.
- Less efficient than other constructions
  - Size of witness in $O(\ln(n))$.

Open Problem

- “Is it possible to build a *strong, dynamic* and *universal* accumulator with witness size lower than $O(\ln(n))$?”
Thank you!
Invoice Factoring using accumulator

- We need a secure broadcast channel
  - If a message \( m \) is published, every participant sees the same \( m \).

- Depending on the security level required
  - Trusted http of ftp server
  - Bulletin Board [CGS97]
Invoice Factoring using accumulator

Factoring Authority

FE_1 FE_2 \ldots FE_i \ldots FE_n

Provider

Client

(1) Invoice
(2) Ack

(3) Invoice, Ack

(4) Is there the invoice?

(5) YES / NO

We need to see in detail this step

(1) Invoice

(2) Ack
Invoice Factoring using accumulator

- Step 5 (Details)

FE

```plaintext
Have you got invoice x?

YES/NO, w_x

If NO, insert x

Check(Acc_{before}, w_x, x)

CheckUpdate(Acc_{before}, Acc_{after}, w_x)

All tests pass => I can buy x.
```

Factoring Authority

```plaintext
w_x = \text{Witness}(m_{before}, x)

Acc_{new, w_{up}} = \text{UpdateAdd}(m_{before, x})
```
Distributed solutions?

- Complex to implement
- Hard to make them robust
- High bandwidth communication
- Need to be online – synchronization problems

That’s why we focus on a centralized solution.
Checking for (non-)membership

User

Belongs(Acc, \( w_x \)) = True \Leftrightarrow x \in X

If \( w_x \) is not valid Belongs returns ⊥.

Accumulator Manager

\( w_x = \text{Witness}(m,x) \)

Memory

Does \( x \) belong to \( X \)?

\[ \bot \]
Update of the accumulated value

User

CheckUpdate(Acc\text{before}, Acc\text{after}, w_{up})

Accumulator Manager

m_{after}, \ Acc_{after}, w_{up} = \text{Update}_{\text{Add/Del}}(m_{before}, x)
How to delete elements?

$X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$

element to be deleted: $x_2$
How to delete elements?
How to delete elements?

\[ \left(-\infty, x_1 \right) \]
\[ (x_1, x_3) \quad (x_6, \infty) \]
\[ (x_5, x_6) \quad (x_3, x_4) \quad (x_4, x_5) \]
Bibliography

- [BeMa92] Efficient Broadcast Time-Stamping *Josh Benaloh and Michael de Mare* 1992
- [BeMa94] One-way Accumulators: A decentralized Alternative to Digital Signatures *Josh Benaloh and Michael de Mare*, 1994
- [BarPfi97] Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees *Niko Barić and Birgit Pfitzmann* 1997
- [Koch98] On certificate revocation and validation *P.C. Kocher* 1998
- [CGH98] The random oracle methodology revisited *R. Canetti, O. Goldreich and S. Halevi* 1998
- [Sand99] Efficient Accumulators Without Trapdoor *Tomas Sanders* 1999
- [GoTa01] An efficient and Distributed Cryptographic Accumulator *Michael T. Goodrich and Roberto Tamassia* 2001
- [CamLys02] Dynamic Accumulators And Application to Efficient Revocation of Anonymous Credentials *Jan Camenisch and Anna Lysyanskaya* 2002
- [GeRa04] RSA Accumulator Based Broadcast Encryption *Craig Gentry and Zulfikar Ramzan* 2004
- [LLX07] Universal Accumulators with Efficient Nonmembership Proofs *Jiangtao Li, Ninghui Li and Rui Xue* 2007
- [AWSM07] Compact E-Cash from Bounded Accumulator *Man Ho Au, Qianhong Wu, Willy Susilo and Yi Mu* 2007
- [WWP08] A new Dynamic Accumulator for Batch Updates *Peishun Wang, Huaxiong Wang and Josef Pieprzyk* 2008
- [CKHO08] Strong Accumulators from Collision-Resistant Hashing *Philippe Camacho, Alejandro Hevia, Marcos Kiwi, and Roberto Opazo* 2008