Project Report on Solving Discrete Logarithm Problems with Auxiliary Input

ECC 2012 @ Queretaro, Mexico
Yumi Sakemi (FUJITSU Laboratories Ltd, Japan)

Joint work with Goichiro Hanaoka (AIST), Tetsuya Izu, Masahiko Takenaka and Masaya Yasuda (FUJITSU Laboratories Ltd.)

October 29th, 2012
ECC and ECDLP

- Elliptic Curve Cryptosystems (ECC)
  - One of the standardized public key cryptosystem

- Elliptic Curve Discrete Logarithm Problem (ECDLP)
  - $G$: a point on an elliptic curve
  - $G = \langle G \rangle$: an additive cyclic group generated by $G$ with prime order $r$
  - ECDLP: a problem to find $x$ from $G$ and $xG$
  - Best algorithms for solving ECDLP are square-root methods.
  - ECDLP is infeasible if $r > 2^{160}$

<table>
<thead>
<tr>
<th></th>
<th>BSGS method [Shanks71]</th>
<th>$\rho$-method [Pollard78]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
<td>$2 \sqrt{r}$</td>
<td>$\sqrt{\pi r/2}$</td>
</tr>
<tr>
<td><strong>Space</strong></td>
<td>$2 \sqrt{r}$</td>
<td>$(\sqrt{\pi r/2})/\theta$</td>
</tr>
<tr>
<td><strong>How to find a solution</strong></td>
<td>Deterministic</td>
<td>Probabilistic</td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td>Easy</td>
<td>Hard</td>
</tr>
<tr>
<td><strong>Storage size</strong></td>
<td>Large</td>
<td>Less</td>
</tr>
</tbody>
</table>
Pairing-based Cryptosystems

- New schemes have been constructed with pairings
  - Identity-based encryption (IBE), broadcast encryption, ...

To assure the security of such cryptosystems, a lot of new mathematical problems have been introduced.
ECDLPwAI (ECDLP with Auxiliary Input)

ECDLP with Auxiliary Input [EUROCRYPT 06]
- Input: $G, xG, x^dG \in G, d|(r-1)$
- Output: $x \in \mathbb{Z}/r\mathbb{Z}$

Cheon’s Algorithm [EUROCRYPT 06]
- An efficient algorithm for solving ECDLPwAI
- Uses BSGS method or $\rho$-method as a subroutine
- Time Complexity: $T = O\left(\sqrt{r/d} + \sqrt{d}\right)$ depending on $d$
  - $d \approx \sqrt{r} \rightarrow T = O\left(4\sqrt{r}\right)$
- Eg: when 160-bit elliptic curve is used,
  - Solving ECDLP requires $2^{80}$
  - Solving ECDLPwAI (by Cheon’s algorithm) requires $\geq 2^{40}$
Implication to pairing-based cryptosystems

Solving ECDLPwAI implies breaking some cryptosystems

DHBDHP

L-BDEP

L-SFP

SXDHP

CDHP

BDHP

L-BDHEP

L-sSDHP

DDHP

DDHIP

L-SDHP

DLINP

DHIP

SDDHIP

WDHP

SDDHIP

Solving ECDLPwAI implies breaking some cryptosystems
Implication to ECDLP

- Cheon’s algorithm can solve ECDLP in ECC which static DH Oracle is available in
  - ElGamal Encryption Scheme [ElGamal84]
  - Ford-Kaliski Key Retrieval Scheme [FK00]

\[
G, \ xG, \ x^{dG}, \ \sqrt{d} + \sqrt{r/d}
\]

The ECDLP can be solved by Cheon’s algorithm in this framework
Motivation of Our Project

- Records for solving mathematical problems
  - For solving ECDLPwAI, there are little experimental results.
  - For solving old mathematical problems, there are many theoretical and experimental results.

<table>
<thead>
<tr>
<th>Mathematical problem</th>
<th>Algorithm</th>
<th>Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factorization Problem</td>
<td>Number Field Sieve</td>
<td>768-bit (Dec, 2009)</td>
</tr>
<tr>
<td>DLP</td>
<td>Function Field Sieve</td>
<td>923-bit (Jun, 2012)</td>
</tr>
<tr>
<td>ECDLP</td>
<td>Square Root Method</td>
<td>112-bit (Jul, 2009)</td>
</tr>
</tbody>
</table>

Motivation

To evaluate the infeasibility of ECDLPwAI by implementing Cheon’s algorithm
Outline of Our Project

Chapter 1
- 60-bit
- 128-bit
- 160-bit

Chapter 2

Chapter 3

Chapter 4

Solved by Jao, Yoshida [Pairing09]
Cheon’s Algorithm

**Input**: $G, G_1 = xG, G_d = x^dG \in G, d|(r-1)$

**Output**: $x \in \mathbb{Z}/r\mathbb{Z}$

**Approach**
- Instead of finding $x$ directly, find an integer $k$ such that $x = \zeta^k$ for a generator $\zeta \in (\mathbb{Z}/r\mathbb{Z})^*$
- Cheon’s algorithm finds $k_1, k_2$ such that $k = k_1 + k_2 (r-1)/d$ in two steps.

**Procedures**

- **Step 1**: Find $k_1$ such that $G_d = \zeta_d^{k_1} G$ (for $\zeta_d = \zeta^d$)
- **Step 2**: Set $e \leftarrow (r-1)/d, G_e \leftarrow \zeta^{-k_1} G_1$
- **Step 3**: Find $k_2$ such that $G_e = \zeta_e^{k_2} G$ for $\zeta_e = \zeta^e$
- Output $x = \zeta^k$ for $k = k_1 + k_2 \times e$
Baby-step Giant-step Method (BSGS method)

**Approach (Shanks, 1971 [Shanks71])**
- Let \( m = \sqrt{r} \), solution \( x = ym + z \ (0 \leq y, z < m) \)
  - \( y \) and \( z \) are uniquely determined
- Let \( G_1 = xG \), then \( G_1 = yG' + zG \Rightarrow y \) and \( z \) satisfy \( G_1 - zG = yG' \)
- Instead of finding \( x \) directory, find \( y \) and \( z \) such that \( G_1 - zG = yG' \)
- Generate following two DBs and found points satisfy \( G_1 - zG = yG' \) by comparing among two DBs
  - Baby-step: \( G_1, G_1-G, G_1-2G, \ldots, G_1-(m-1)G \)
  - Giant-step: \( 0, G', 2G', \ldots, (m-1)G' \)

**Complexity**
- Time complexity: \( 2\sqrt{r} \)
- Space Complexity: \( 2\sqrt{r} \)
Baby-step Giant-step Method

- Find $x = 96$ (unknown) from $G$ and $xG$ (with $r = 100$)
- Exhaustive search requires 50 evaluations (on average)
- BSGS method only requires at most 20 evaluations

$x = 6 + 9 \times 10 = 96$

Baby-step ($xG - iG$)

Giant-step ($j \times 10G$)

$xG = iG + j \times \sqrt{r} G$

(0 $\leq$ $i$, $j$ $<$ $\sqrt{r}$)

Collision
Applying BSGS Method on Step 1

- Set $G_d \leftarrow x^d G$, $d_1 = \left\lceil \sqrt{(r-1)/d} \right\rceil$, $0 \leq u_1, v_1 < d_1$
- Find $k_1$ such that $\zeta_d^{-u_1} G_d = \zeta_d^{v_1 \times d_1} G$ for $k_1 = u_1 + v_1 \times d_1$
Kozaki-Kutsuma-Matsuo’s method [KKM07]

- Cheon’s algorithm iterates scalar multiplications for same point
  - In the case of ECDLP, KKM method is not required
    ⇒ next point is calculated by only one group operation

$$sP = s_0P + s_1nP + s_2n^2P + s_3n^3P = T(s_0,0)+T(s_1,1)+T(s_2,2)+T(s_3,3)$$

\[
\begin{array}{|c|c|c|c|}
\hline
T(i,j) & 1 & 2 & \cdots & n-1 \\
\hline
0 & P & 2P & \cdots & (n-1)P \\
1 & nP & 2nP & \cdots & (n-1)nP \\
2 & n^2P & 2n^2P & \cdots & (n-1)n^2P \\
3 & n^3P & 2n^3P & \cdots & (n-1)n^3P \\
\hline
\end{array}
\]

(O( log r )-group operations)
Comparison between two DBs

- After generating points, two DBs need to be compared.

Baby-step

<table>
<thead>
<tr>
<th>Index</th>
<th>Element Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>0x0001</td>
<td></td>
</tr>
<tr>
<td>0x0002</td>
<td></td>
</tr>
<tr>
<td>0x0003</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x(d1-1)</td>
<td></td>
</tr>
</tbody>
</table>

Giant-step

<table>
<thead>
<tr>
<th>Index</th>
<th>Element Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>0x0001</td>
<td></td>
</tr>
<tr>
<td>0x0002</td>
<td></td>
</tr>
<tr>
<td>0x0003</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x(d1-1)</td>
<td></td>
</tr>
</tbody>
</table>

Collision!! k1
Our Approach: Bucket-sorting

\[ \# = 2^{(\log d_1)/2^6} \]

- Our approach requires \( O( n \log n/64 ) \) \( \approx (\log d_1) \log (\log d_1)/64 \)
- Parallel procession is possible

Upper 8-bit in Element Data

...
Target Elliptic Curve Parameters

- **Target**: TinyTate library [TinyTate07]
  - Pairing-based cryptographic library for embedded devices

### Parameters

- **$E/GF(p^{256})$**: $y^2 = x^3 + x$ (Supersingular)
  - $p^{256} = 3778160688\ 9598235856\ 7455764726\ 5839472148\ 1625071533\ 3029839574\ 7614203820\ 7746163$ (256-bit)
  - $#E = 3778160688\ 9598235856\ 7455764726\ 5839472148\ 1625071533\ 3029839574\ 7614203820\ 7746164$ (256-bit)
  - $r = 1701411885\ 3107163264\ 4604909702\ 696927233$ (128-bit)

- $d = 1268213655\ 0675316736$ (64-bit) ← Ideal for adversaries
- $d_1 = 3662760472$ (32-bit)
- $d_2 = 3561198752$ (32-bit)
1 PC is used
### Experimental Results: BSGS method

<table>
<thead>
<tr>
<th></th>
<th>Required Time</th>
<th>Required Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>65.6 hours</td>
<td>246 Gbyte</td>
</tr>
<tr>
<td>Step 2</td>
<td>65.8 hours</td>
<td>246 Gbyte</td>
</tr>
<tr>
<td>Total</td>
<td>131.4 hours</td>
<td>max 246 Gbyte</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel® Core™ i7 2.93 GHz (4 cores)</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 10.04</td>
</tr>
<tr>
<td>Language</td>
<td>C</td>
</tr>
<tr>
<td>Library</td>
<td>GMP</td>
</tr>
</tbody>
</table>
## Estimations: BSGS method

<table>
<thead>
<tr>
<th>r</th>
<th>DB Generation</th>
<th>Collision Search</th>
<th>Total</th>
<th>Required storage size</th>
</tr>
</thead>
<tbody>
<tr>
<td>128-bit</td>
<td>6 days</td>
<td>17 hours</td>
<td>7 days</td>
<td>288 GB</td>
</tr>
<tr>
<td>132-bit</td>
<td>11 days</td>
<td>35 hours</td>
<td>13 days</td>
<td>576 GB</td>
</tr>
<tr>
<td>136-bit</td>
<td>22 days</td>
<td>74 hours</td>
<td>25 days</td>
<td>1152 GB</td>
</tr>
<tr>
<td>140-bit</td>
<td>45 days</td>
<td>6 days</td>
<td>51 days</td>
<td>2304 GB</td>
</tr>
</tbody>
</table>

- Required times is not problematic when r becomes larger.
- However, required storage size will be beyond 2TB if r is 140-bit.
- In such a case, BSGS method will not work.
Chapter 2

Chapter 1

Chapter 4

Solved by Jao, Yoshida [Pairing09]
Step1 of Cheon’s algorithm

- Find partial solution $k_1$ such that $G_d = \zeta_d^{k_1}G$, $\zeta_d = \zeta^d$

Approach ([Pollard78])

- Instead of finding $k_1$, find $u$ and $v$ such that $\zeta_d^uG_d = \zeta_d^vG$ anyway
  - $\Rightarrow$ since $\zeta_d^{k_1}G = \zeta_d^{v-u}G$, $k_1 = v-u$
  - $u$ and $v$ can be probabilistically found by generating $\zeta_d^iG_d$ and $\zeta_d^jG$ by random-walk (RW) function $F$
    - $F: \zeta_d^{i+1}P = F(\zeta_d^iP)$
  - Notice
    - a collision such that $\zeta_d^{i1}G_d = \zeta_d^{i2}G_d$ or $\zeta_d^{i1}G = \zeta_d^{i2}G$ can not find $u$ and $v$
    - In ECDLP, such a collision can find the solutions

Complexity (based on the birthday paradox)

- Time complexity: $2 \sqrt{\pi d}$
- Space Complexity: $\sqrt{\pi d}$
Applying $\rho$-method on Step 1

- For obtaining $k_1$, step 1 finds $u$ and $v$ such that $\zeta_d^u G_d = \zeta_d^v G_d$

- From an initial point $G_d$, points $\zeta_d^u G_d$ are randomly generated by using the random-walk function $F(G_d)$
Applying $\rho$-method on Step 1 (cont’d)

- For obtaining $k_1$, step 1 finds $u$ and $v$ such that $\zeta_d^u G_d = \zeta_d^v G$

- Similarly, from an initial point $G$, points are $\zeta_d^i G$ randomly generated by using the random-walk function $F(G)$
Applying $\rho$-method on Step 1 (cont’d)

If we find collided points such that $\zeta_d^{uG_d} = \zeta_d^{vG}$ in blue points and red points, the partial solution value $k1=v-u$ is obtained.
Distinguished element technique

Idea

- Only store distinguished element in DB
  - Distinguished element: the x coordinates is divisible by a certain integer $\theta$
  - Space complexity can be reduced to $\left(\sqrt{\pi d/2}\right)/\theta$

- There exists collisions on distinguished elements
Problems in $\rho$-method

- Problem 1: RW function $F(G_d)$ outputs same points in DB for $G_d$
- Problem 2: RW function $F(G_d)$ does not output next points

These problems also exist in RW function $F(G)$

Since the number of points does not increase we can not find a collision
Causes of Problems

Since RW function randomly generates points, a collision is occurred in points generated by $G_d$

- Problem 1: a collision occurred in distinguished elements generated by $G_d$
- Problem 2: a collision occurred in undistinguished elements generated by $G_d$

This collision is called fruitless cycle

We can not find a collision
Required features for $\rho$-method

If a collision in points by $G_d$ is detected, reset the initial point

- **How to detect**
  - **Problem1:** If RW function $F(G_d)$ outputs same point, a collision in distinguished elements by $G_d$ can be detected
  - **Problem2:** If RW function $F(G_d)$ does not output in some intervals, fruitless cycle can be detected
Target Elliptic Curve Parameters

- **Target:** TinyTate library [TinyTate07]
  - Pairing-based cryptographic library for embedded devices

- **Parameters**
  - \( E/GF(p256) : y^2 = x^3 + x \) (Supersingular)
  - \( p_{256} = 3778160688 \ 9598235856 \ 7455764726 \ 5839472148 \ 1625071533 \ 3029839574 \ 7614203820 \ 7746163 \) (256-bit)
  - \( \#E = 3778160688 \ 9598235856 \ 7455764726 \ 5839472148 \ 1625071533 \ 3029839574 \ 7614203820 \ 7746164 \) (256-bit)
  - \( r = 1701411885 \ 3107163264 \ 4604909702 \ 696927233 \) (128-bit)
  - \( d = 1268213655 \ 0675316736 \) (64-bit) ← Ideal for adversaries
  - \( \theta = 2^{18} \)

Same as 1st experiment
### Experimental Results: $\rho$-method

<table>
<thead>
<tr>
<th></th>
<th>Required Time (single core)</th>
<th>Required Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>68.2 hours</td>
<td>0.5 Mbyte</td>
</tr>
<tr>
<td>Step 2</td>
<td>67.2 hours</td>
<td>0.5 Mbyte</td>
</tr>
<tr>
<td>Total</td>
<td>135.4 hours</td>
<td>max 0.5 Mbyte</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel® Core™ i7 2.93 GHz (4 cores)</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 10.04</td>
</tr>
<tr>
<td>Language</td>
<td>C</td>
</tr>
<tr>
<td>Library</td>
<td>GMP</td>
</tr>
</tbody>
</table>
## Comparison with BSGS method

<table>
<thead>
<tr>
<th></th>
<th>Required Time of $\rho$-method</th>
<th>Required Time of BSGS method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1</strong></td>
<td>68.2 hours</td>
<td>65.6 hours</td>
</tr>
<tr>
<td><strong>Step 2</strong></td>
<td>67.2 hours</td>
<td>65.8 hours</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>135.4 hours</td>
<td>131.4 hours</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Required storage size of $\rho$-method</th>
<th>Required storage size of BSGS method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1</strong></td>
<td>0.5 M byte</td>
<td>246 G byte</td>
</tr>
<tr>
<td><strong>Step 2</strong></td>
<td>0.5 M byte</td>
<td>246 G byte</td>
</tr>
</tbody>
</table>
Difficulties of solving 160-bit ECDLPwAI

- For solving 160-bit ECDLPwAI, time complexity become too large
  - Estimated cost becomes 1,440 days on a single core

Solving system for large scale parameters
- Parallelization
- High scalability
  - Many PCs are easily added to the system
Parallelization

$G_d$

$\zeta_d G_d$

$\zeta^d \ell G_d$

DB#1

collision!!

comparing

k1

DB#2

$\xi_d G$

$\xi^d \ell G$

Copyright 2012 FUJITSU LABORATORIES Ltd.
Problem in Parallelization

Problem: a collision is occurred in one side of DB

⇒ If detect a collision in one side of DB, reset the initial points

We can not find a collision
Grand Design of the Solving System

(Everyone can access)

Server

The Internet

Proxy

Labotary “A”

Clients...

Proxy

Univ. “B”

Clients...

Proxy

Cloud Computing resources

Clients...

Copyright 2012 FUJITSU LABORATORIES Ltd.
Construction of the solving system

- Parallelization of RW functions is available
- It is easy to add clients
- Our system can be used for solving ECDLP
Target Elliptic Curve Parameters

- BN(Barreto-Naehrig) curve [BN05]
  - Ordinary pairing-friendly elliptic curve
  - $G$: 160-bit prime order $r$
    - $r = 146150162449679026514544738099497$
      - 118849930027613
    - $d = 2 \times 3 \times 12132793 \times 135993458106516349$ (84-bit)
    - $(r-1)/d = 2 \times 164442871007 \times 448873741399$ (77-bit)
Our System for Solving ECDLPwAI

- Although any computing resources can be joined to the system, but we only used our PCs.
Experimental Result

We have successfully solved ECDLPwAI on an elliptic curve with 160-bit in 25 days (1,314 core days)

<table>
<thead>
<tr>
<th>Step</th>
<th>CPU (Hz)</th>
<th># of PCs</th>
<th># of cores</th>
<th>Time [days]</th>
<th>core × days</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Q9450 (2.66GHz)</td>
<td>8</td>
<td>32</td>
<td>7</td>
<td>224</td>
</tr>
<tr>
<td></td>
<td>Q9450 (3.00GHz)</td>
<td>8</td>
<td>32</td>
<td>13</td>
<td>416</td>
</tr>
<tr>
<td>2</td>
<td>X3460 (2.80GHz)</td>
<td>10</td>
<td>80</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Pentium D (3.40GHz)</td>
<td>9</td>
<td>18</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>subtotal</td>
<td>max 162</td>
<td></td>
<td></td>
<td></td>
<td>1090</td>
</tr>
</tbody>
</table>

**Total**

1314

*Hyper-Threading is used*
Chapter 4

BSGS

160-bit

128-bit

60-bit

r

???-bit

Solved by Jao, Yoshida [Pairing09]

Chapter 2

Chapter 3

Chapter 4
Monetary Cost Estimation

- Cost for solving ECDLPwAI of 160-bit is 1,314 core × days ⇒ $2,810 in Amazon EC2

- If we can invest $1,000,000 ⇒ ECDLPwAI of 204 bit order is solved

Amazon EC2 becomes cheaper :-)

In PKC2012, it was estimated to 192-bit
If we can use super computer K …

160-bit ECDLPwAI $\Rightarrow$ 1 minutes!!
Impact on cryptographic schemes

- Boneh, Gentry, and Waters’ Broadcast Encryption Scheme [BGW05]
  - The security of this scheme is assured by the infeasibility of $L$-BDHE problem

- Key Construction
  - Secret key: $x \in \mathbb{Z}/r\mathbb{Z}$ is a random number
  - Public key: $pk = (G, xG, \ldots, x^L G, x^{L+2} G, \ldots, x^{2L} G, V)$ ($L$ : # of receiver)

- Applying Cheon’s algorithm
  - If $d \leq 2L$, Cheon’s algorithm can be applied to $L$-BDHE problem
160-bit curve case

- \( \log_2 d \)
- \( \log_2 r \)

Selectable range of parameter \( d \)

# of receiver should be chosen smaller than \( 2^{80} \)

Solvable range within $2,810

Solved
160-bit curve case

- Selectable range of parameter $d$
- # of receiver should be chosen smaller than $2^{58}$

Solved

- Solvable range within $2,810$
- Solvable range within $1,000,000$
Feedback to protocol with static DH Oracle

- E.g: 160-bit elliptic curve
- Since our system is available for Cheon’s algorithm whose complexity is $2^{41}$, the number of oracle calls becomes $2^{80}$
- It is difficult to execute $2^{80}$ oracle calls

![Diagram](attachment:image.png)

**Feedback to protocol with static DH Oracle**

- E.g: 160-bit elliptic curve
- Since our system is available for Cheon’s algorithm whose complexity is $2^{41}$, the number of oracle calls becomes $2^{80}$
- It is difficult to execute $2^{80}$ oracle calls
Feedback to protocol with static DH Oracle

- E.g: 160-bit elliptic curve
- Cheon’s algorithm whose complexity is $2^{51}$ can be executed with $1,000,000$
- The number of queries can be reduced to $d=2^{58}$

In this framework, the 160-bit ECDLP could be solved in the not-so distant future.
## Summary of Our Project

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Character</th>
<th>Size of $r$</th>
<th>BSGS (1 core)</th>
<th>$\rho$ (1 core)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yao-Yoshida [Pairing 09]</td>
<td>$p$</td>
<td>60-bit</td>
<td>–</td>
<td>3 hours</td>
</tr>
<tr>
<td>Izu-Takenaka-Yasuda [WAIS 10/AINA 11]</td>
<td>3</td>
<td>83-bit</td>
<td>14 hours</td>
<td>11.5 hours</td>
</tr>
<tr>
<td>S-Izu-Takenaka-Yasuda [WISTP 11/WISA11]</td>
<td>$p$</td>
<td>128-bit</td>
<td>131.4 hours</td>
<td>135.4 hours</td>
</tr>
<tr>
<td>S-Hanaoka-Izu-Takenaka-Yasuda [PKC 12]</td>
<td>$p$</td>
<td>160-bit</td>
<td>–</td>
<td>1314 days (25 actual days)</td>
</tr>
</tbody>
</table>
Future Works

- Compare the fruitless cycles of $\rho$-method for ECDLP and that for ECDLPwAI
- Evaluate securities of pairing-based cryptographies with other mathematical problems experimentally
- Brake the record for solving ECDLP by using our solving system
shaping tomorrow with you


References


References