Implementation and evaluation of a group encryption scheme

Bachelor’s Thesis Presentation

Peter Ten

Advisor: M.Sc. Christoph Hagen

5/2/2020

https://se.informatik.uni-wuerzburg.de
WHAT IS MY THESIS ABOUT?
What is my thesis about?

➢ Explanation of Nishat’s scheme
➢ Implementation of the scheme
➢ Verify correctness
➢ Test practicability (performance)

We want end-to-end encryption for groups

One public key associated with a group

Most current group-oriented encryption are too static
  - Problems with key management
  - Rekeying causes overhead

Forward secrecy is desirable

No rekeying of secret keys desirable
GROUP-ORIENTED ENCRYPTION
<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ A group consists of a specific set of users</td>
<td>➢ A group is associated with a public key</td>
<td>➢ Messages can be encrypted and sent to the group</td>
<td>➢ Group members can decrypt the message</td>
</tr>
</tbody>
</table>

Implementation and evaluation of a dynamic group-oriented encryption scheme

P. Ten
Nishat’s scheme (properties)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ A dynamic scheme with constant rekeying cost</td>
<td>➢ Constant public and private key sizes</td>
<td>➢ Constant ciphertext size</td>
<td>➢ Individual secret keys</td>
</tr>
<tr>
<td>➢ Secret keys remain the same after group changes</td>
<td>➢ Forward and backward secrecy</td>
<td>➢ (Fast performance and low storage cost)</td>
<td></td>
</tr>
</tbody>
</table>
Nishat’s scheme (group creation)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>creates Group, creates public key</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>private key for each user</td>
</tr>
</tbody>
</table>

Implementation and evaluation of a dynamic group-oriented encryption scheme

P. Ten
### Nishat’s scheme (member add)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>computes new public key</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>private keys don't change</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>gets a private key</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Idea:**
- Computes new public key
- Private keys don't change

**Actions:**
- Gets a private key
Nishat’s scheme (member leave)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>computes new public key</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>private keys don't change</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HOW DOES NISHAT’S SCHEME WORK?
Nishat’s scheme (functionality)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

- Split public and private keys into sub-keys
- Public number to calculate a secret for decryption → **public parameter**

\[
PK_A = (PK_{A1}, PK_{A2}), \quad PK_{A1} = g^{\alpha k} \quad \text{and} \quad PK_{A2} = g^{\beta k}
\]

\[
SK_i = (s_{i1}, s_{i2}, s_{i3}, s_{i4}, s_{i5}, s_{i6}, s_{i7}), \quad s_{i7} = r_i
\]

Public parameter: \( \gamma = (a \times r_1 \times \ldots \times r_n) - k \)

\( \alpha, \beta, k, a, r_i \in \mathbb{Z}_p^*, \quad k < r_i \quad \forall i \in \{1, \ldots, n\} \)
### Nishat’s scheme (functionality)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ A user can calculate $k$ with help of $\gamma$</td>
<td>➢ A member uses $k$ and ${s_{i1}, \ldots, s_{i6}}$ to decrypt a message</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Public parameter: $\gamma = (a \times r_1 \times \ldots \times r_n) - k$

$$\kappa = r_i - \gamma \mod r_i$$
### Nishat’s scheme (functionality)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ The administrator can easily add and remove a user</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Addition:**

\[
\gamma' = (\gamma + k) \times a^{-1} \times a' \times r_{n+1} - k'
\]

**Removal:**

\[
\gamma' = (\gamma + k) \times a^{-1} \times a' \times r^{-1} - k'
\]

PK'\_A = (PK'\_A\_1, PK'\_A\_2), PK'\_A\_1 = g^{\alpha k'} and PK'\_A\_2 = g^{\beta k'}
<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

**WHAT ARE THE PROBLEMS?**
## Nishat’s scheme (problems)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typing error in crucial formula</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Public parameter cannot be closed</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Implementation and evaluation of a dynamic group-oriented encryption scheme  

*P. Ten*
### Typing error in crucial formula:

**Original formula:**
\[
\bar{\xi}_2 = \frac{s_{i3} \cdot (s_{i4})^{\kappa - 1}}{(s_{i5})^{\kappa - 1}}
\]

**Corrected formula:**
\[
\bar{\xi}_2 = \frac{s_{i3} \cdot (s_{i4})^{\kappa}}{(s_{i5})^{\kappa - 1}}
\]
### Nishat’s scheme (problems)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

Public parameter cannot be closed:

$$\gamma = (a \times r_1 \times \ldots \times r_n) - k$$

$$\kappa = r_i - \gamma \mod r_i$$

$$k, a, r_i \in \mathbb{Z}_p^*, \ k < r_i \ \forall i \in \{1, \ldots, n\}$$
Nishat’s scheme (problems)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

\[ \kappa = r_i - \gamma \mod r_i \]

\[ = r_i - ((a \times r_1 \times \ldots \times r_n) - k) \mod r_i \]

\[ = r_i - ((a \times r_1 \times \ldots \times r_n) \mod r_i - (k \mod r_i)) \]

\[ = r_i - (0 - k) \mod r_i \]

\[ = r_i - (-k \mod r_i) \]

\[ = r_i - (r_i - k) \]

\[ = k \]
**Nishat’s scheme (problems)**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

Proof with counterexample:

\[ \mathbb{Z}_p^* = \mathbb{Z}_{13}^*, \quad a = 3, \quad r_1 = 5, \quad r_2 = 7, \quad k = 2 \]

Public parameter:

\[ \gamma = (((3 \cdot 5 \cdot 7) \mod 13) - 2) \mod 13 \]

\[ = ((105 \mod 13) - 2) \mod 13 \]

\[ = (1 - 2) \mod 13 \]

\[ = 12 \]
Nishat’s scheme (problems)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

Proof with counterexample:

\[ \mathbb{Z}_p^* = \mathbb{Z}_{13}^*, \ a = 3, \ r_1 = 5, \ r_2 = 7, \ k = 2 \]

try to calculate \( \kappa \) and \( r_1 \):

\[ \kappa = (5 - 12 \mod 5) \mod 13 \]

\[ = 5 \mod 13 - 2 \mod 13 \]

\[ = (5 - 2) \mod 13 \]

\[ = 3 \neq k = 2 \]

⇒ use "usual" multiplication in a field instead of ring operations
### Nishat’s scheme (problems)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition:</td>
<td>$\gamma' = (\gamma + k) \times a^{-1} \times a' \times r_{n+1} - k'$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Removal:</td>
<td>$\gamma' = (\gamma + k) \times a^{-1} \times a' \times r_{x}^{-1} - k'$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\Rightarrow a^{-1} = \frac{1}{a}$, $r_{x}^{-1} = \frac{1}{r_{x}}$
- Dynamic encryption scheme for groups
- Secret keys do not change
- Constant private key size
- Constant ciphertext size
- Fast encryption and decryption
- Still large amount of use cases
- Easy group management
<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verified correctness</td>
<td>Evaluated performance</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Implementation and evaluation of a dynamic group-oriented encryption scheme

P. Ten
Implementation and evaluation of a dynamic group-oriented encryption scheme

P. Ten
Actions (implementation)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

```
typedef struct groupAdmin {
    mpz_t order;          // for internal calculations
    element_t alpha;      // part of master secret key
    element_t beta;       // part of master secret key
    element_t betaInv;    // for internal calculations
    element_t gToAlpha;   // part of system parameter
    element_t gToBeta;    // part of system parameter
    mpz_t smallest_ri;    // smallest ri for calculating k
    mpz_t k;              // current k
    mpz_t a;              // current a
} groupAdmin_t;
```

```
typedef struct user {
    element_t s1;          // sub-key of private key
    element_t s2;          // sub-key of private key
    element_t s3;          // sub-key of private key
    element_t s4;          // sub-key of private key
    element_t s5;          // sub-key of private key
    element_t s6;          // sub-key of private key
    element_t s7;          // sub-key of private key
    int id;                // unique id
} user_t;
```
### Implementation and evaluation of a dynamic group-oriented encryption scheme

**P. Ten**

#### Actions (implementation)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Algorithm: group setup and key generation

**Input:** headNode, groupAdmin, groupEntity  
**Output:** void

1. Initialize curve and structs
2. Choose random $g \in G_1$ and update groupEntity
3. Choose random $h \in G_1$ and update groupEntity
4. Choose random $\alpha \in \mathbb{Z}_p^*$ and update groupAdmin
5. Choose random $\beta \in \mathbb{Z}_p^*$ and update groupAdmin
6. Calculate $g^\alpha$ and update groupAdmin
7. Calculate $g^\beta$ and update groupAdmin
8. **for all users in user list do**
    9. Calculate secret key like in the scheme and update user  
        **if current s7 < smallest_ri then**
        10. Set new smallest_ri = s7 and update groupAdmin
11. firstPublicKeyParam(headNode, groupAdmin, groupEntity)  
    // Calculate the public key and public parameter like in the scheme  
    and update groupEntity
<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

- Test environment of authors: Dual Core Intel Pentium 3 GHz x 2, 4 GB RAM on Ubuntu 14.04 LTS
- My test environment: Quad Core Intel Haswell i7 4790k 4GHz x 4 (4,4GHz Turbo Boost), 16 GB RAM on Windows 10 64-bit
Implementation and evaluation of a dynamic group-oriented encryption scheme

P. Ten

**Actions (performance)**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

Storage cost:

\[ n = \text{amount of users} \]

<table>
<thead>
<tr>
<th></th>
<th>Author's measurement</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator</td>
<td>( O(n) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>User</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>Public key</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>Ciphertext</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>Public parameter</td>
<td>( O(1) )</td>
<td>( O(n) )</td>
</tr>
</tbody>
</table>
Computation cost:

- Each test 20 times and calculated the average
- For group size up to 300 users, each test was repeated six times
- From group size from 400 up to 1000 users, each test was repeated three times
- For comparison single calculation for group sizes of 10.000, 50.000 and 100.000 users
Actions (performance)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

**Diagram:**

- **space consumption for public parameter**
  - Y-axis: size in kBytes
  - X-axis: amount of users

**Graph:**

- Blue line: data point
- Orange line: standard deviation

Implementation and evaluation of a dynamic group-oriented encryption scheme

*P. Ten*
Implementation and evaluation of a dynamic group-oriented encryption scheme

P. Ten
Actions (performance)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

Encryption:

API: **8,38 milliseconds**, standard deviation of 0,017 milliseconds
Decryption:

API: **15.63 milliseconds**, with standard deviation of 0.02 milliseconds
Public key generation:

**Actions (performance)**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

**Public key generation:**

![Graph showing the total time taken for public key generation vs. number of users. The x-axis represents the number of users ranging from $1 \times 10^5$ to $1 \times 10^6$, and the y-axis represents the total time taken in seconds on a log scale. The graph shows a constant time for different numbers of users.]
### Implementation and evaluation of a dynamic group-oriented encryption scheme

**P. Ten**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Graph: Public Key Generation (Large Scale)**

- Time in milliseconds on the y-axis
- Amount of users on the x-axis (1 to 10,000,000)
- Blue line: Data point
- Orange line: Standard deviation
Private key generation:

![Graph showing total time taken in seconds for different numbers of users]
Implementation and evaluation of a dynamic group-oriented encryption scheme

P. Ten
Implementation and evaluation of a dynamic group-oriented encryption scheme

P. Ten
### Actions (performance)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

**Group changes:**

![Graph showing total time taken (seconds) vs. number of users (n). The graph is a horizontal line indicating a constant time taken regardless of the number of users.](image-url)
Actions (performance)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

![Graph showing member addition time in milliseconds vs amount of users](image)

Implementation and evaluation of a dynamic group-oriented encryption scheme

*P. Ten*
### Actions (performance)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

*Graph:*

**Member addition (large scale)**

- **Y-axis:** Time in milliseconds
- **X-axis:** Amount of users (0 to 100,000)

---

**Implementation and evaluation of a dynamic group-oriented encryption scheme**

*P. Ten*
Implementation and evaluation of a dynamic group-oriented encryption scheme

P. Ten
### Actions (performance)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

![Graph: member removal (large scale)](image)

- **time in milliseconds**
- **amount of users**

Implementation and evaluation of a dynamic group-oriented encryption scheme

*P. Ten*
Actions (performance)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

Implementation and evaluation of a dynamic group-oriented encryption scheme

P. Ten
## Actions (performance)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Idea</th>
<th>Benefit</th>
<th>Actions</th>
</tr>
</thead>
</table>

### Group setup (large scale)

<table>
<thead>
<tr>
<th>amount of users</th>
<th>time in minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10000</td>
<td>15</td>
</tr>
<tr>
<td>20000</td>
<td>20</td>
</tr>
<tr>
<td>30000</td>
<td>25</td>
</tr>
<tr>
<td>50000</td>
<td>30</td>
</tr>
<tr>
<td>60000</td>
<td>35</td>
</tr>
<tr>
<td>70000</td>
<td>40</td>
</tr>
<tr>
<td>80000</td>
<td>45</td>
</tr>
<tr>
<td>90000</td>
<td>50</td>
</tr>
<tr>
<td>100000</td>
<td>55</td>
</tr>
</tbody>
</table>

---

Implementation and evaluation of a dynamic group-oriented encryption scheme

*P. Ten*
Summary

➢ **Problem:** Need for dynamic group-encryption

➢ **Idea:** Nishat’s dynamic group encryption scheme

➢ **Benefit:** True dynamic group encryption scheme

➢ **Action:** Implementation and evaluation of proposed scheme