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)

“Group-oriented encryption for dynamic groups with constant rekeying cost“ by Koti Nishat and B. R. Purushothama (released 2016)

Problem

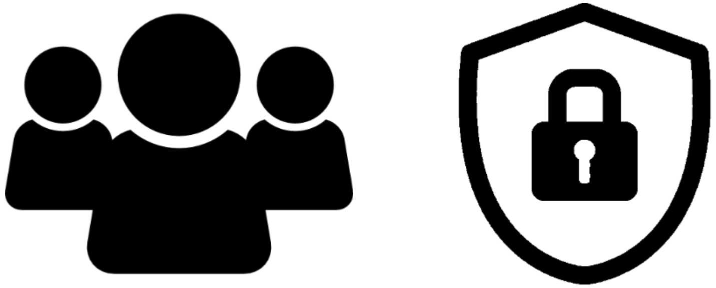
Problem ❯ Idea ❯ Benefit ❯ Actions

- 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

Need for a dynamic group-oriented encryption scheme

Idea

Problem › **Idea** › Benefit › Actions



GROUP-ORIENTED ENCRYPTION

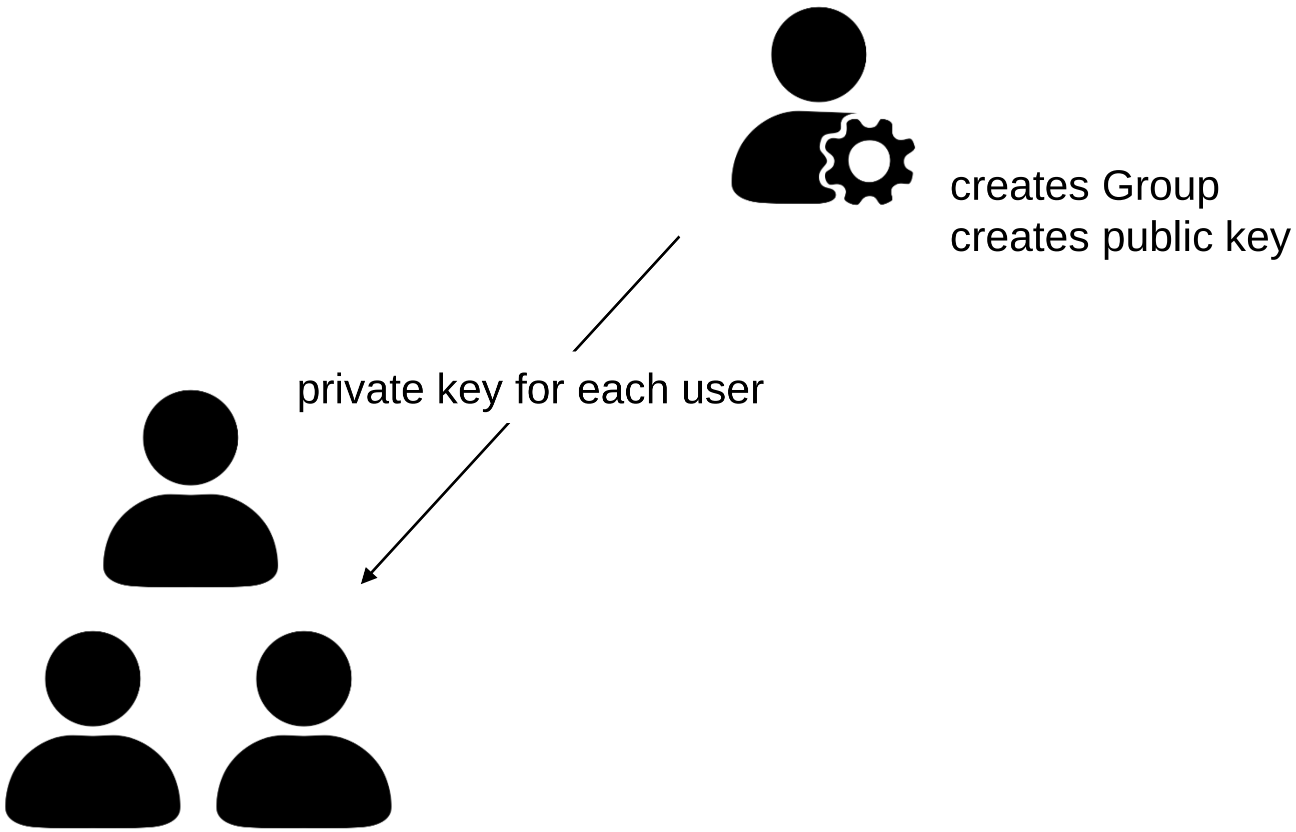
Group-oriented encryption

- A group consists of a specific set of users
- A group is associated with a public key
- Messages can be encrypted and sent to the group
- Group members can decrypt the message

Nishat’s scheme (properties)

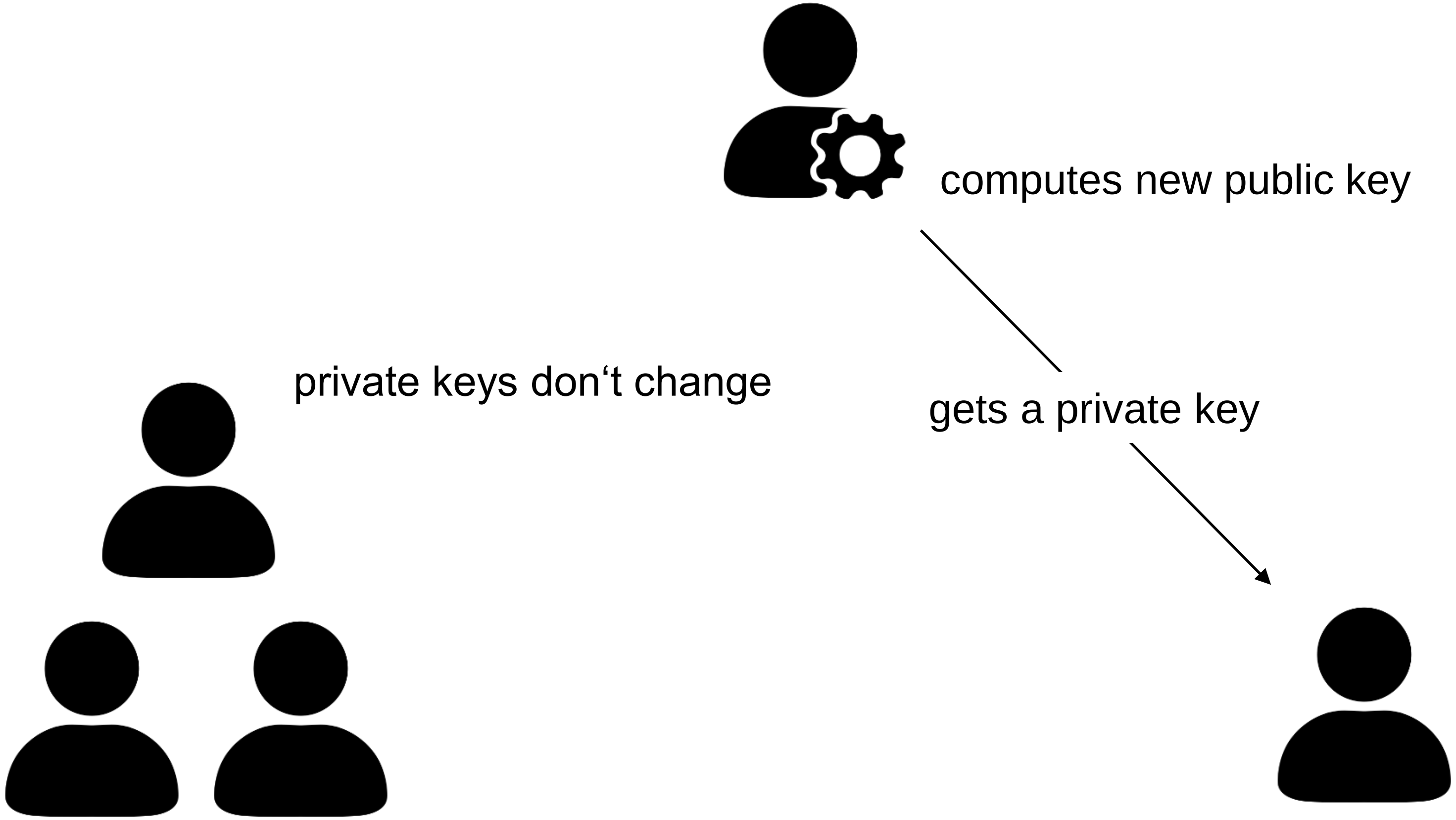
- A dynamic scheme with constant rekeying cost
- Constant public and private key sizes
- Constant ciphertext size
- Individual secret keys
- Secret keys remain the same after group changes
- Forward and backward secrecy
- (Fast performance and low storage cost)

Nishat's scheme (group creation)

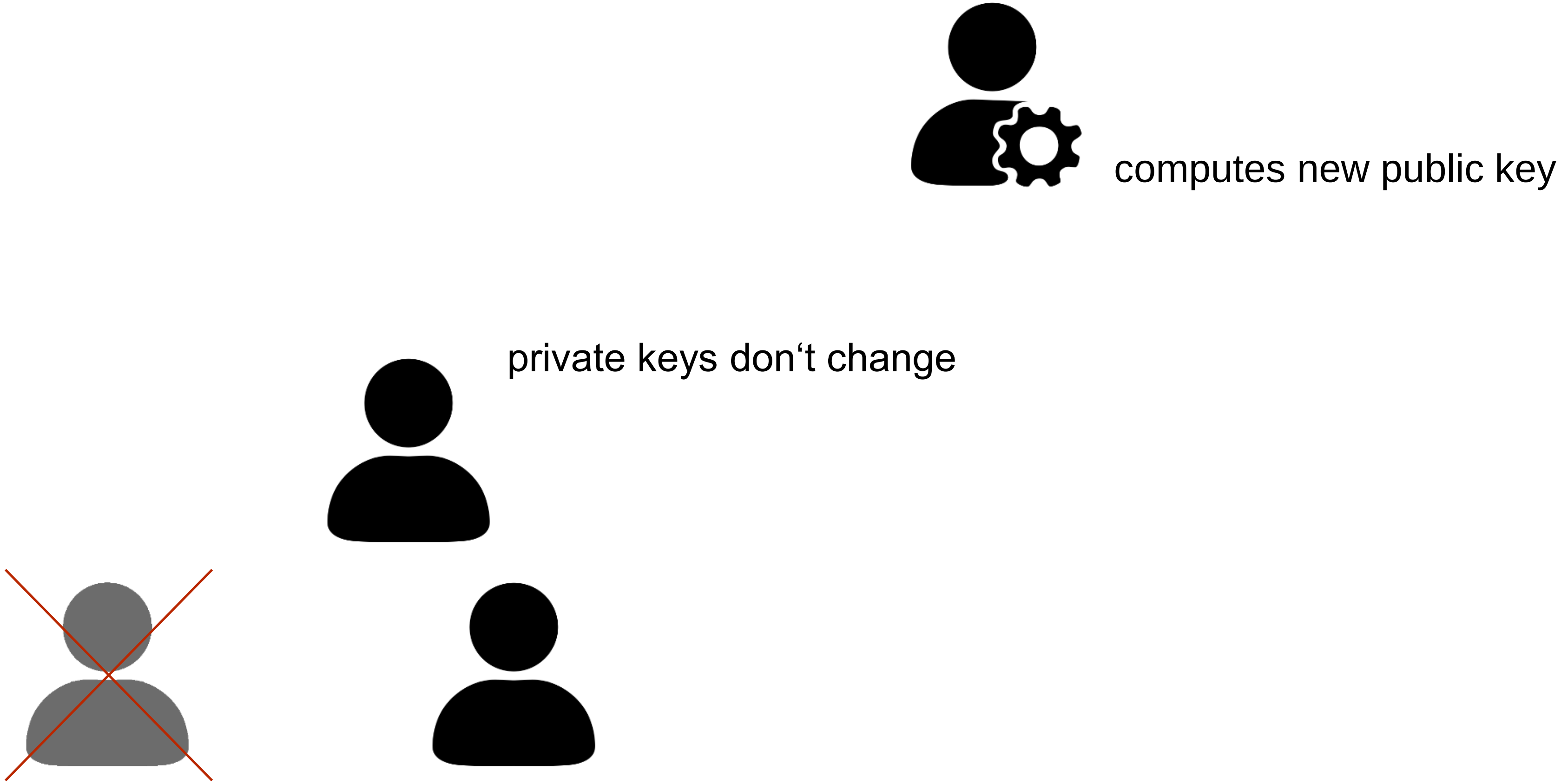


Nishat’s scheme (member add)

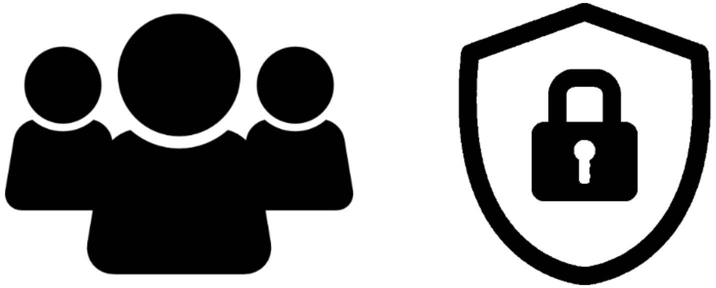
Problem › **Idea** › Benefit › Actions



Nishat’s scheme (member leave)



Nishat‘s scheme (functionality)



HOW DOES NISHAT’S SCHEME WORK?

Nishat's scheme (functionality)

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

$PK_A = (PK_{A_1}, PK_{A_2})$, $PK_{A_1} = g^{\alpha k}$ and $PK_{A_2} = g^{\beta k}$

$SK_i = (s_{i_1}, s_{i_2}, s_{i_3}, s_{i_4}, s_{i_5}, s_{i_6}, s_{i_7})$, $s_{i_7} = r_i$

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

$\alpha, \beta, k, a, r_i \in \mathbb{Z}_p^*$, $k < r_i \; \forall i \in \{1, \ldots, n\}$

Nishat's scheme (functionality)

- A user can calculate k with help of γ
- A member uses k and $\{s_{i_1}, \ldots, s_{i_6}\}$ to decrypt a message

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

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

Nishat’s scheme (functionality)

- The adminstrator can easily add and remove a user

Addition:

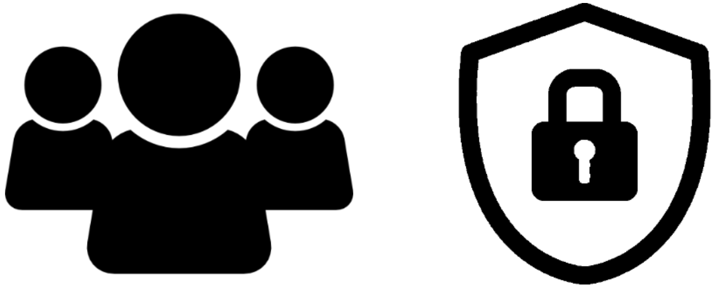
$$\gamma' = (\gamma + k) \times a^{-1} \times a' \times r_{n+1} - k'$$

Removal:

$$\gamma' = (\gamma + k) \times a^{-1} \times a' \times r_x^{-1} - k'$$

$$PK'_A = (PK'_{A_1}, PK'_{A_2}),\ PK'_{A_1} = g^{\alpha k'} \text{ and } PK'_{A_2} = g^{\beta k'}$$

Nishat‘s scheme (problems)



WHAT ARE THE PROBLEMS?

Nishat’s scheme (problems)

- Typing error in crucial formula
- Public parameter cannot be closed

Nishat's scheme (problems)

Problem > **Idea** > Benefit > Actions

Typing error in crucial formula:

original formula: $\xi_2 = \frac{s_{i3} \cdot (si_4)^{\kappa -1}}{(si_5)^{\kappa -1}}$

corrected formula: $\xi_2 = \frac{s_{i3} \cdot (si_4)^{\kappa}}{(si_5)^{\kappa -1}}$

Nishat's scheme (problems)

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)

$$
\begin{aligned}
\kappa &= r_i - \gamma \; mod \; r_i \\
&= r_i - ((a \times r_1 \times \ldots \mathrm{X}\, r_n) - k) \; mod \; r_i \\
&= r_i - ((a \times r_1 \times \ldots \mathrm{X}\, r_n) \; mod \; r_i - (k \; mod \; r_i)) \\
&= r_i - (0 - k) \; mod \; r_i \\
&= r_i - (-k \; mod \; r_i) \\
&= r_i - (r_i - k) \\
&= k
\end{aligned}
$$

Nishat's scheme (problems)

Proof with counterexample:

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

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)

Proof with counterexample:

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

try to calculate κ and r_1:

$$\kappa = (5 - 12 \; mod \; 5) \; mod \; 13$$
$$= 5 \; mod \; 13 - 2 \; mod \; 13$$
$$= (5 - 2) \bmod 13$$
$$= 3 \neq k = 2$$

⇒ use "usual" multiplication in a field instead of ring operations

⇒

Nishat's scheme (problems)

Problem › **Idea** › Benefit › Actions

Addition:

$$\gamma' = (\gamma + k) \text{ x } a^{-1} \text{ x } a' \text{ x } r_{n+1} - k'$$

Removal:

$$\gamma' = (\gamma + k) \text{ x } a^{-1} \text{ x } a' \text{ x } r_x^{-1} - k'$$

$\Rightarrow a^{-1} = \frac{1}{a}, r_x^{-1 =} = \frac{1}{r_x}$

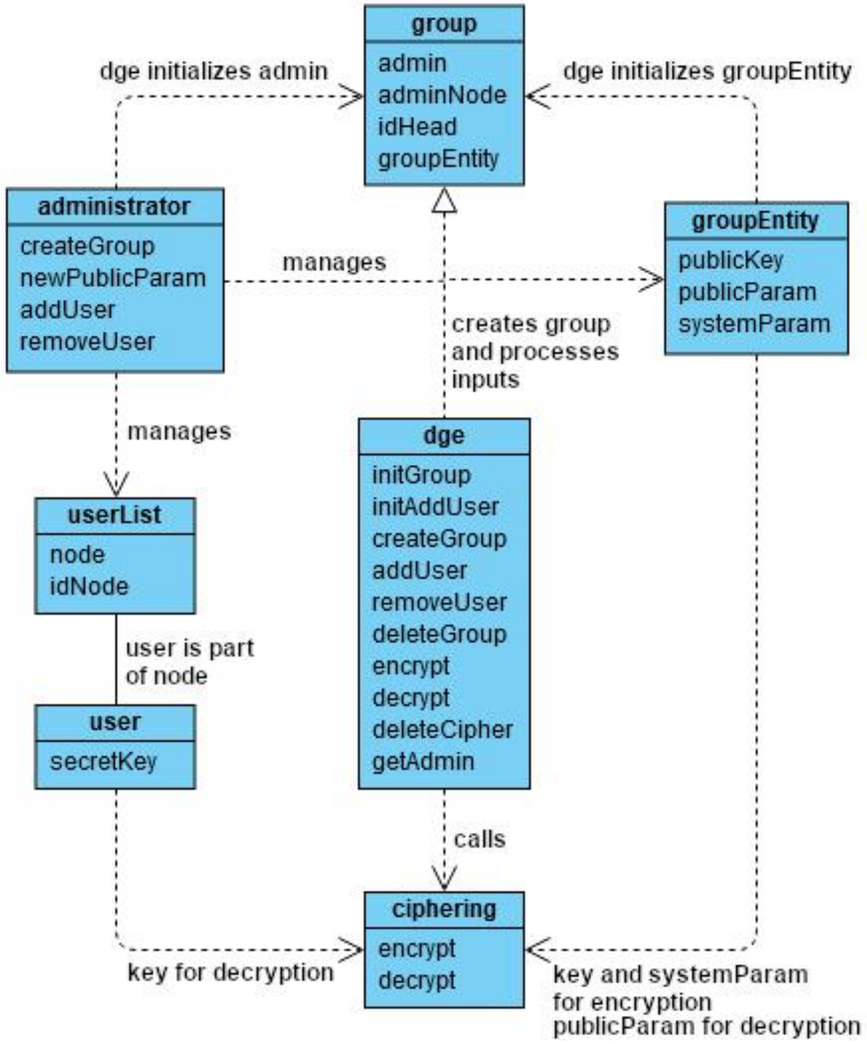
Benefit

- 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

Actions

- Verified correctness
- Evaluated performance

Actions (implementation)



Actions (implementation)

Problem › Idea › Benefit › **Actions**

```
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;
```

Actions (implementation)

Algorithm group setup and key generation

Input: headNode, groupAdmin, groupEntity
Output: void

Initialize curve and structs
Choose random $g \in \mathbb{G}_1$ and update groupEntity
Choose random $h \in \mathbb{G}_1$ and update groupEntity
Choose random $\alpha \in \mathbb{Z}_p^*$ and update groupAdmin
Choose random $\beta \in \mathbb{Z}_p^*$ and update groupAdmin
Calculate g^α and update groupAdmin
Calculate g^β and update groupAdmin
for *all users in user list* **do**
    Calculate secret key like in the scheme and update user
    if *current s7 < smallest_ri* **then**
        Set new smallest_ri = s7 and update groupAdmin
firstPublicKeyParam(headNode, groupAdmin, groupEntity)
`// Calculate the public key and public parameter like in the scheme and update groupEntity`

Actions (performance)

- 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

Actions (performance)

Storage cost:

n = amount of users

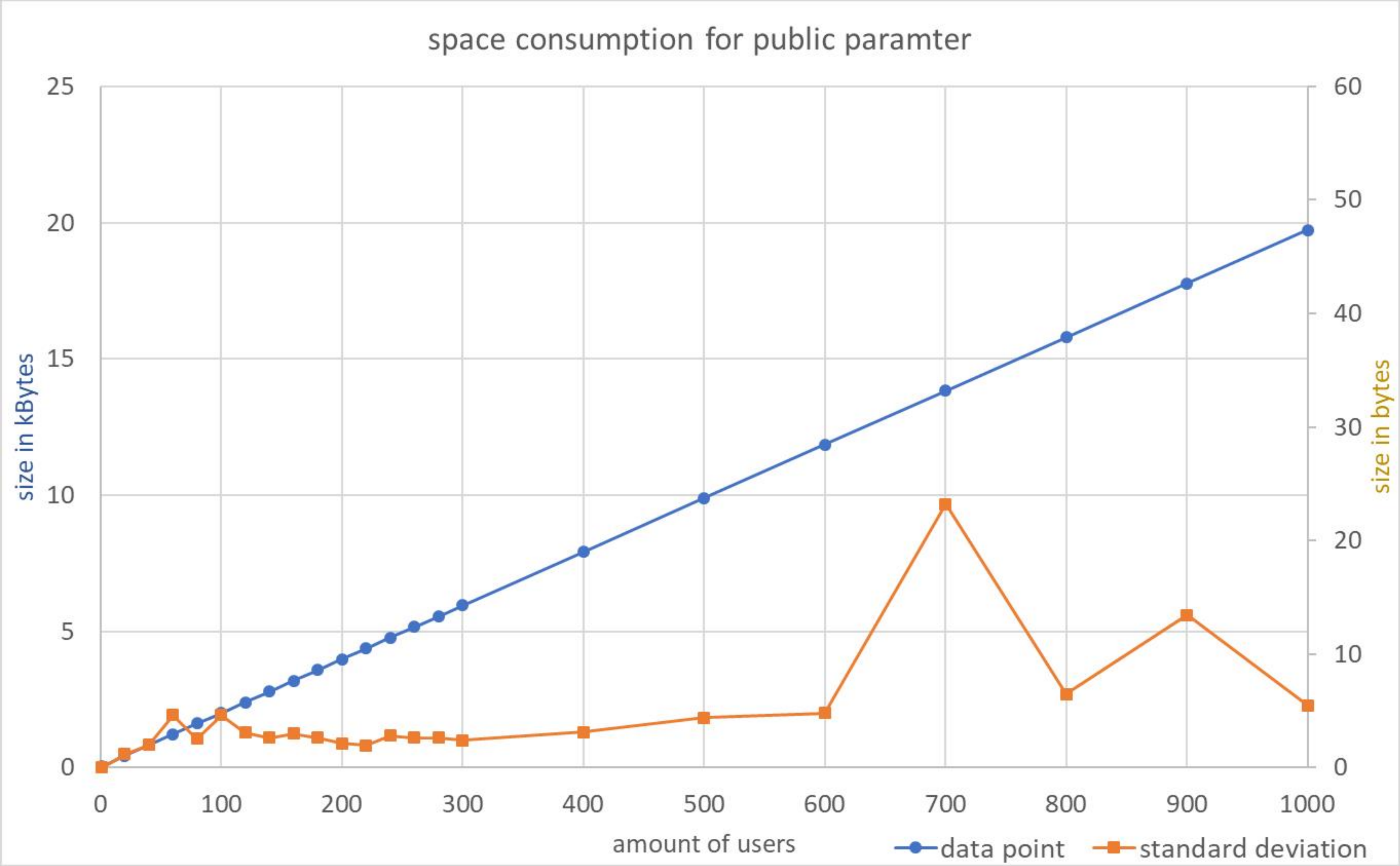
	Author‘s measurement	API
Administrator	O(n)	O(n)
User	O(1)	O(1)
Public key	O(1)	O(1)
Ciphertext	O(1)	O(1)
Public parameter	O(1)	O(n)

Actions (performance)

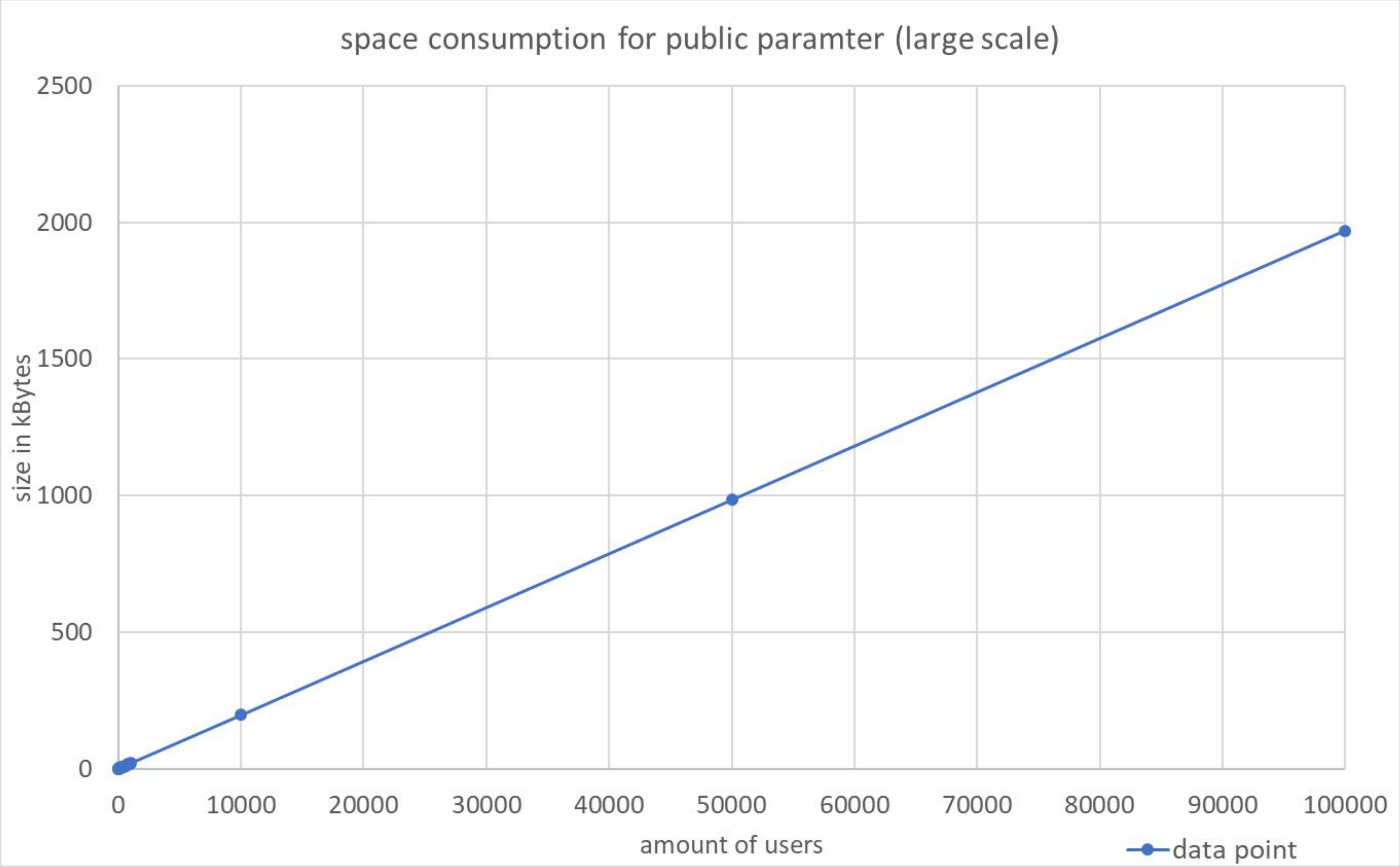
Computation cost:

- 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

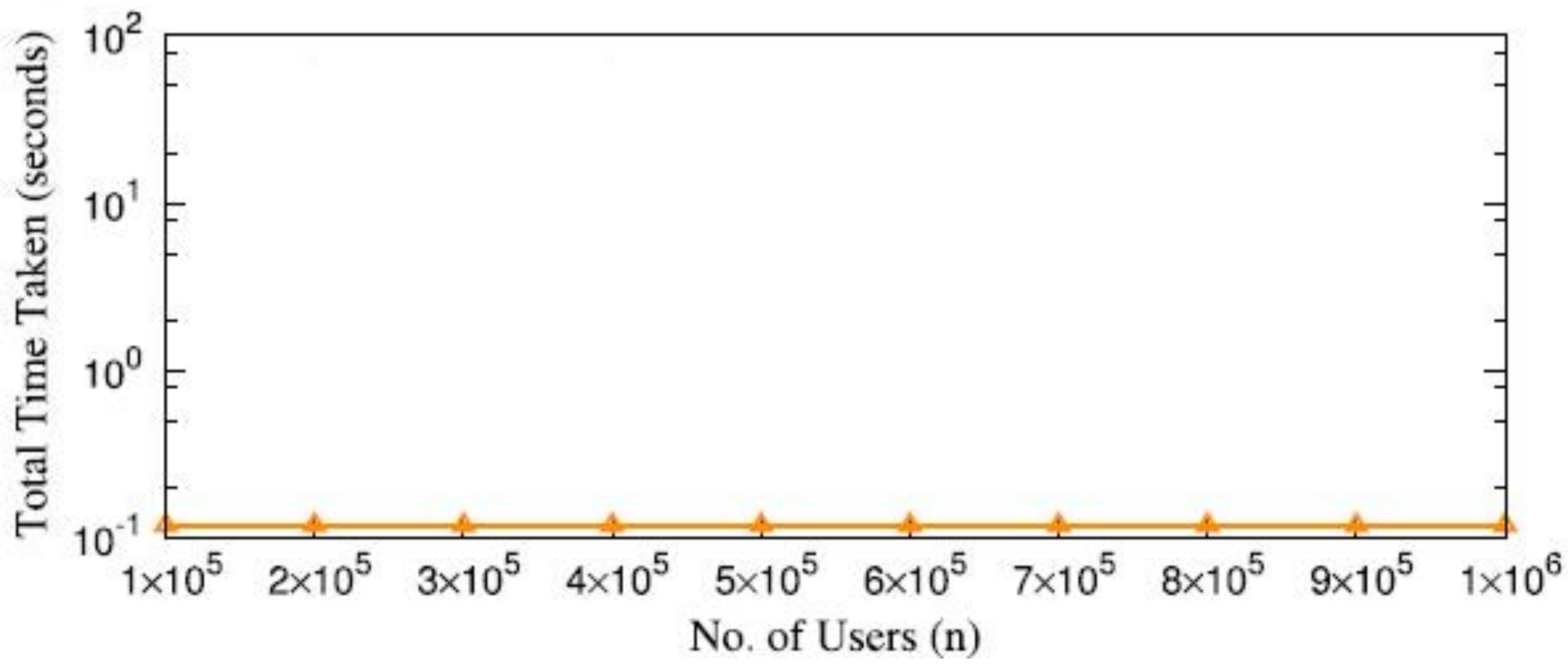
Actions (performance)



Actions (performance)



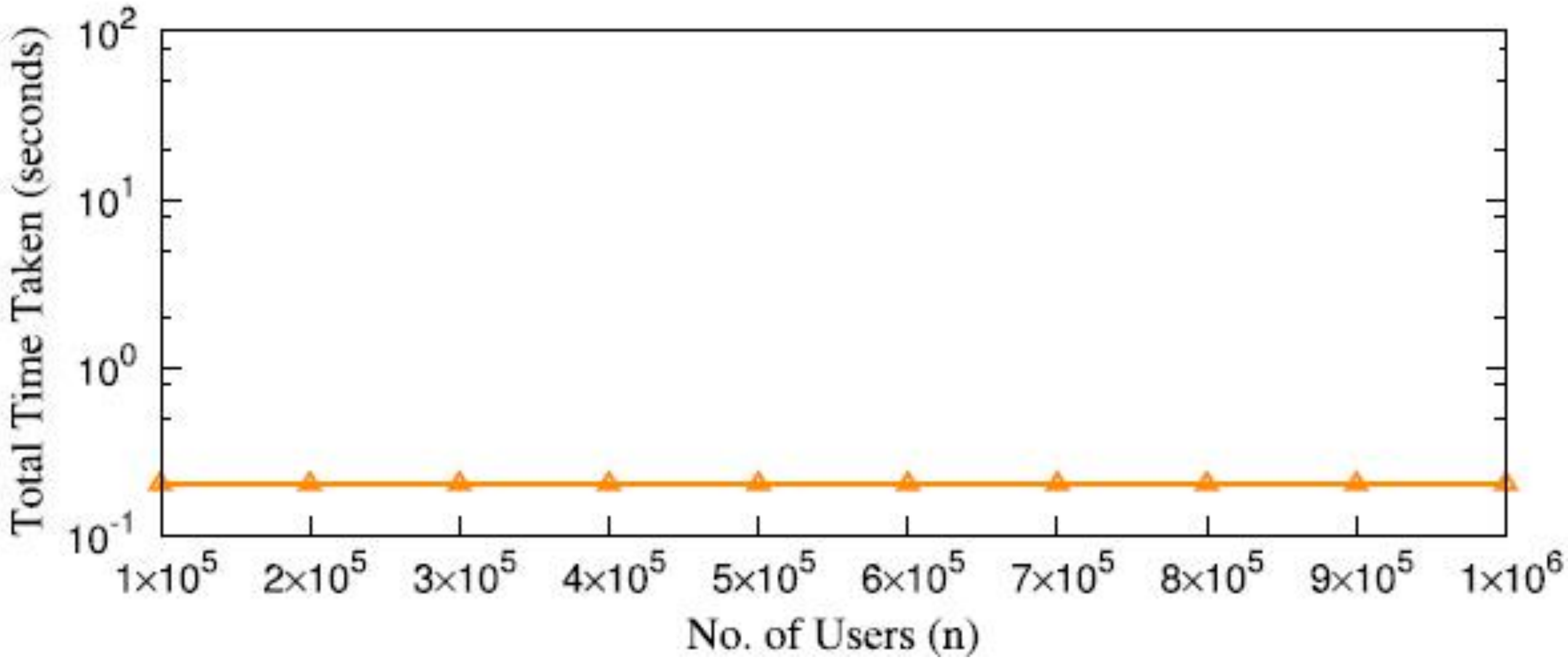
Encryption:



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

Actions (performance)

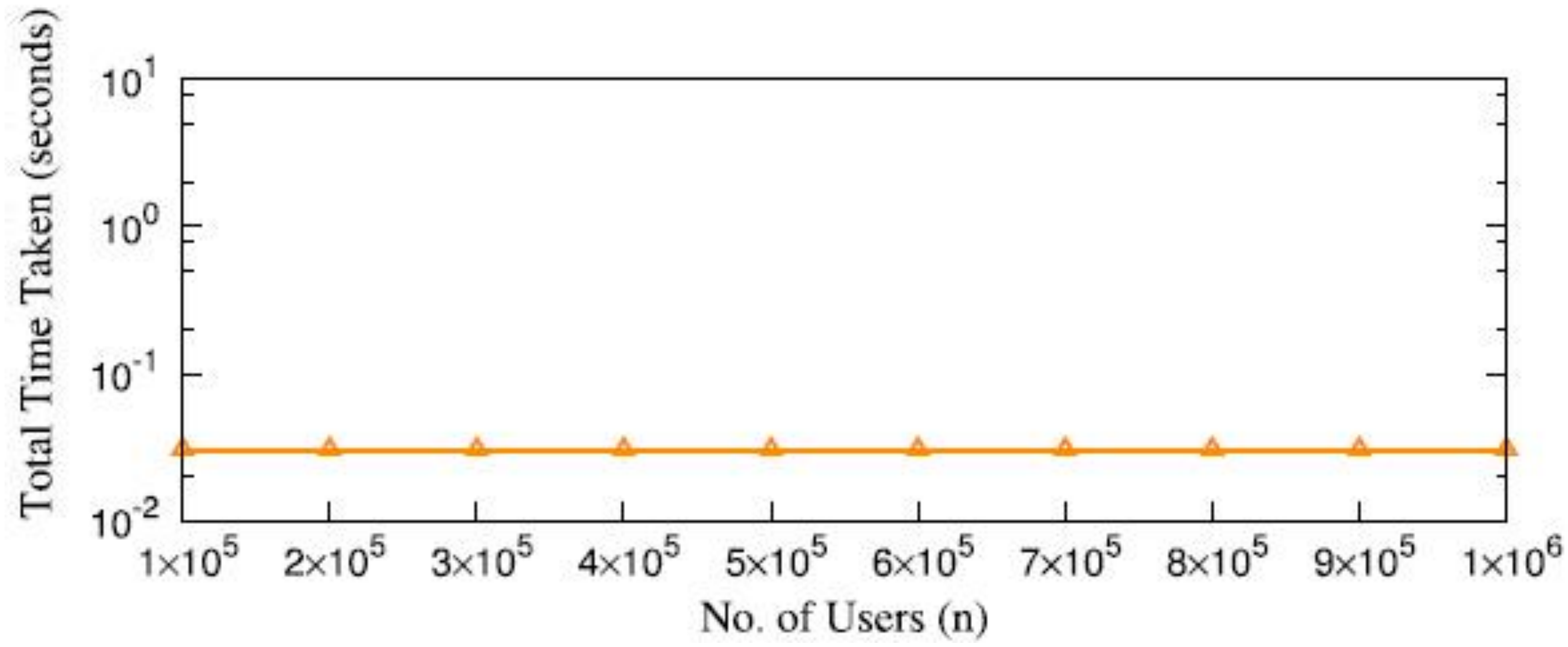
Decryption:



API: **15,63 milliseconds**, with standard deviation of 0,02 milliseconds

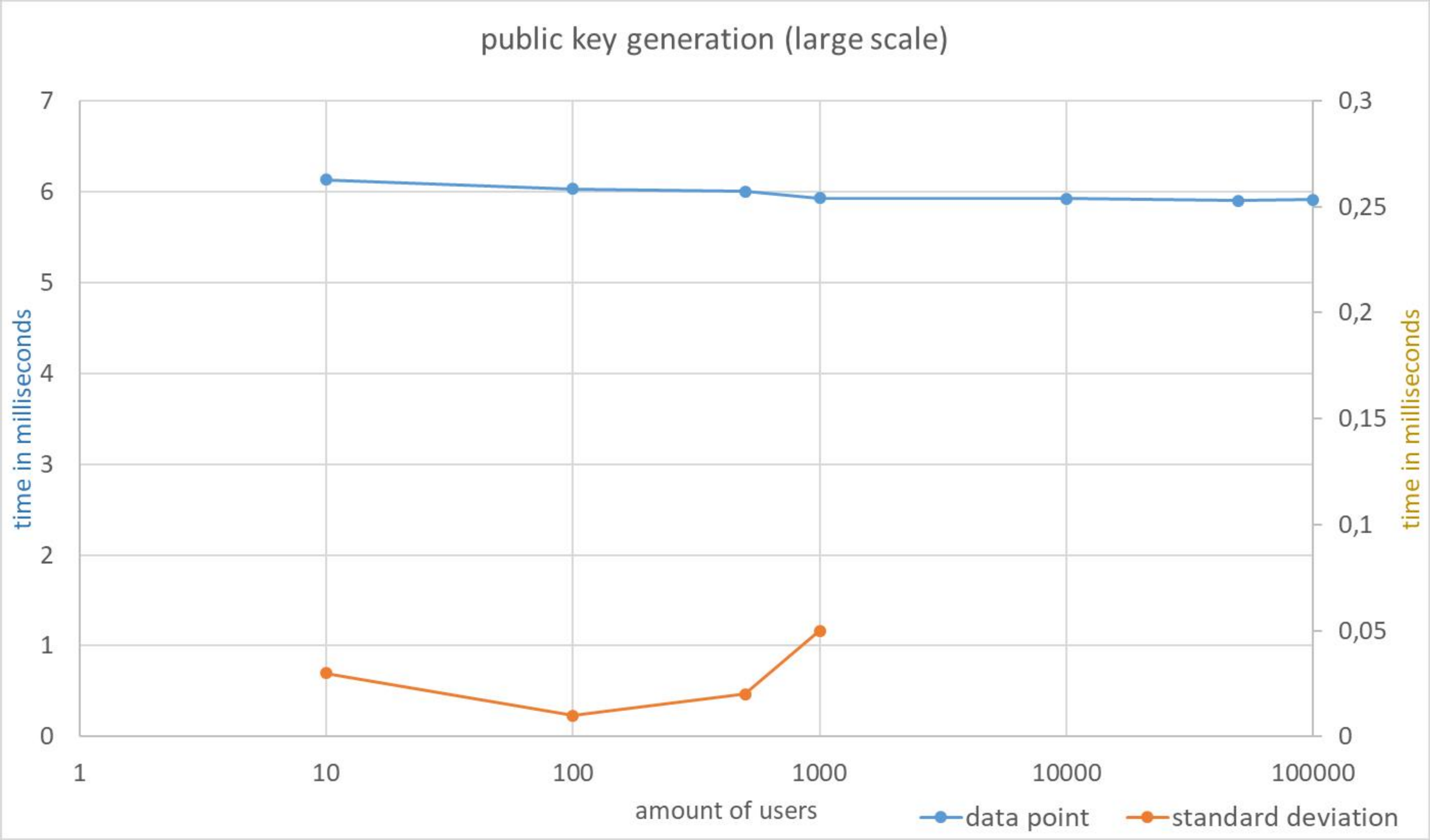
Actions (performance)

Public key generation:



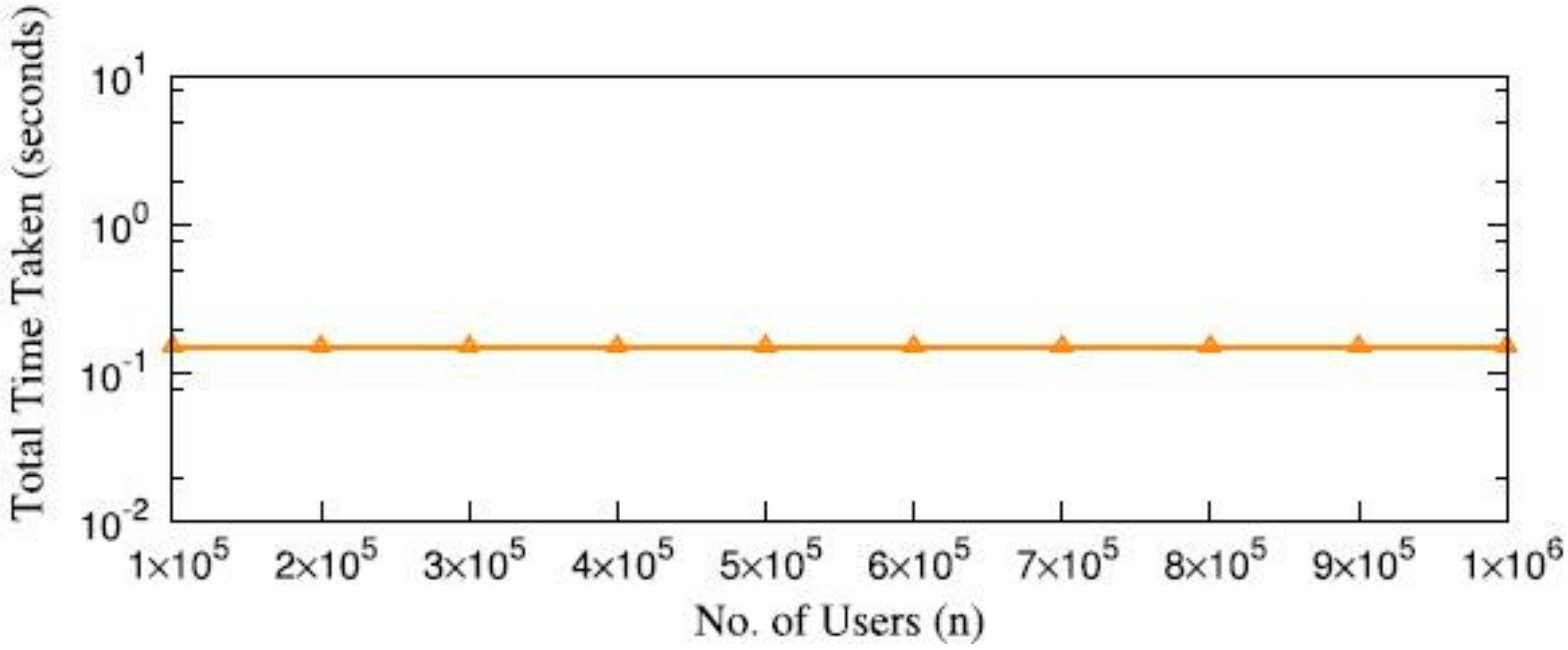
Actions (performance)

Problem › Idea › Benefit › **Actions**



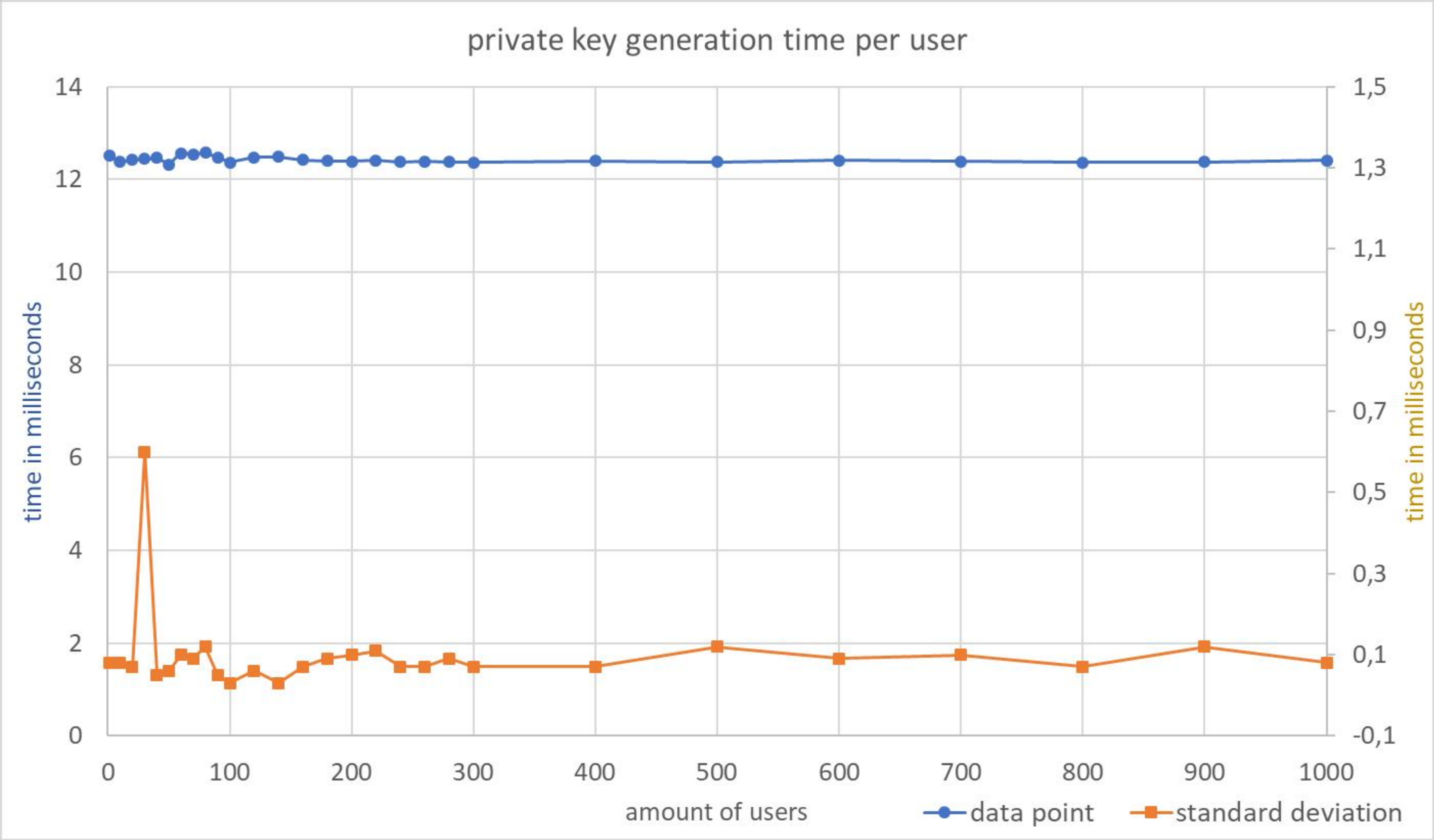
Actions (performance)

Private key generation:



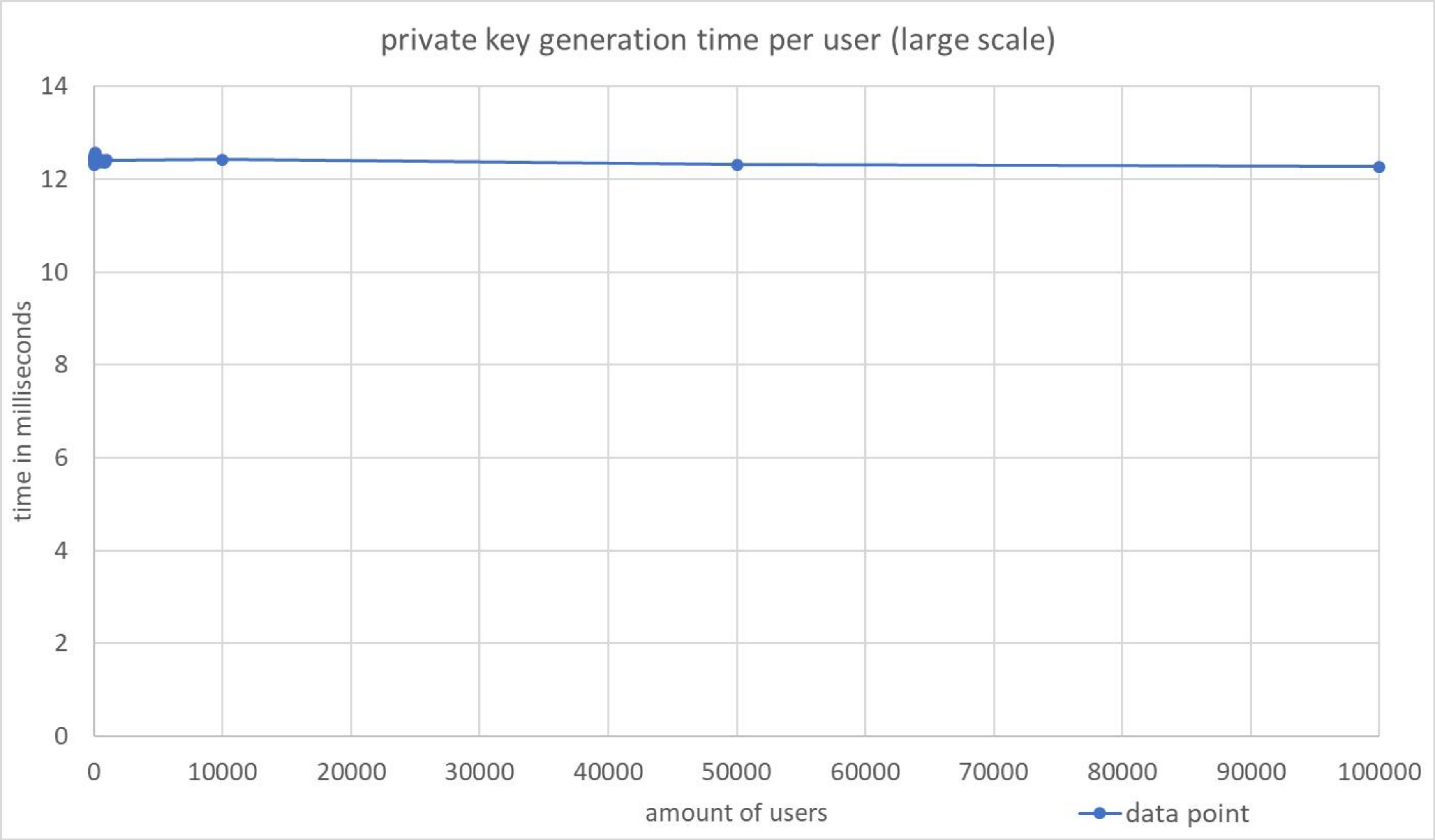
Actions (performance)

Problem › Idea › Benefit › **Actions**



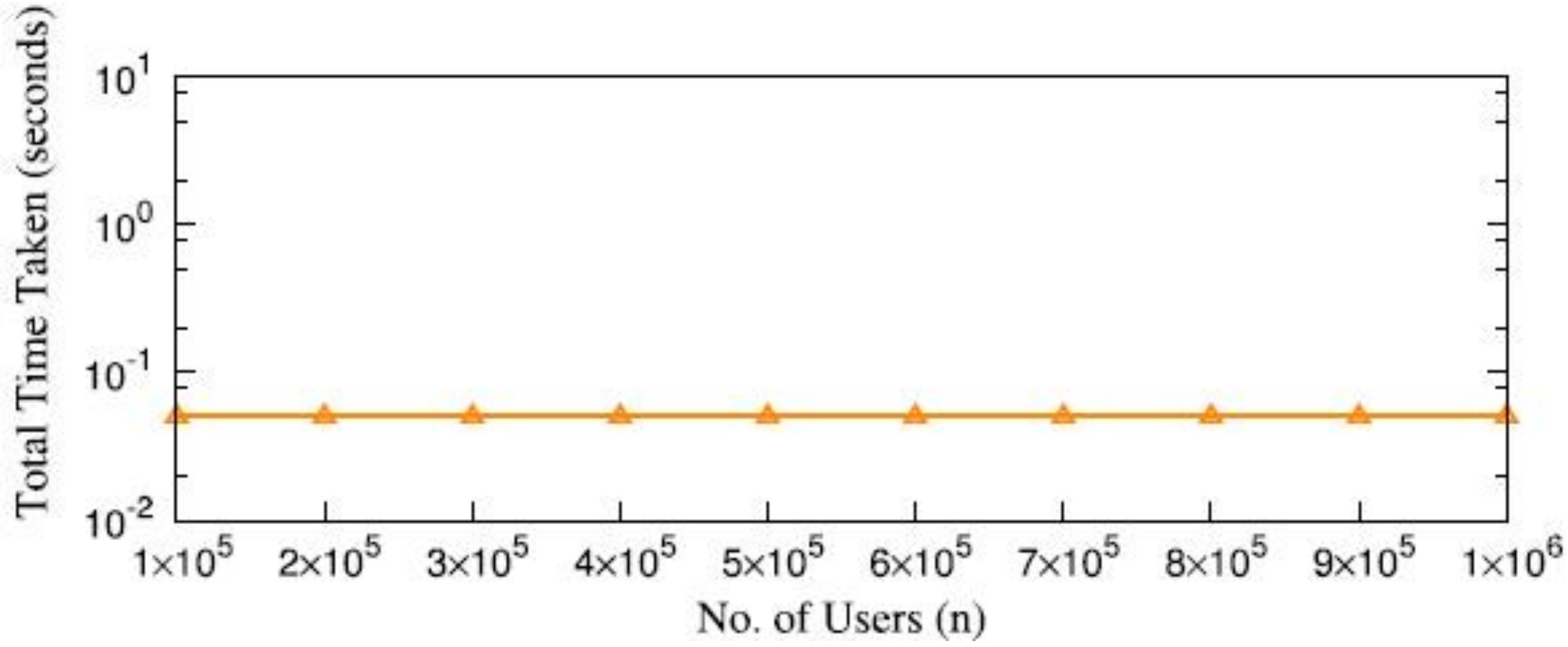
Actions (performance)

Problem › Idea › Benefit › **Actions**

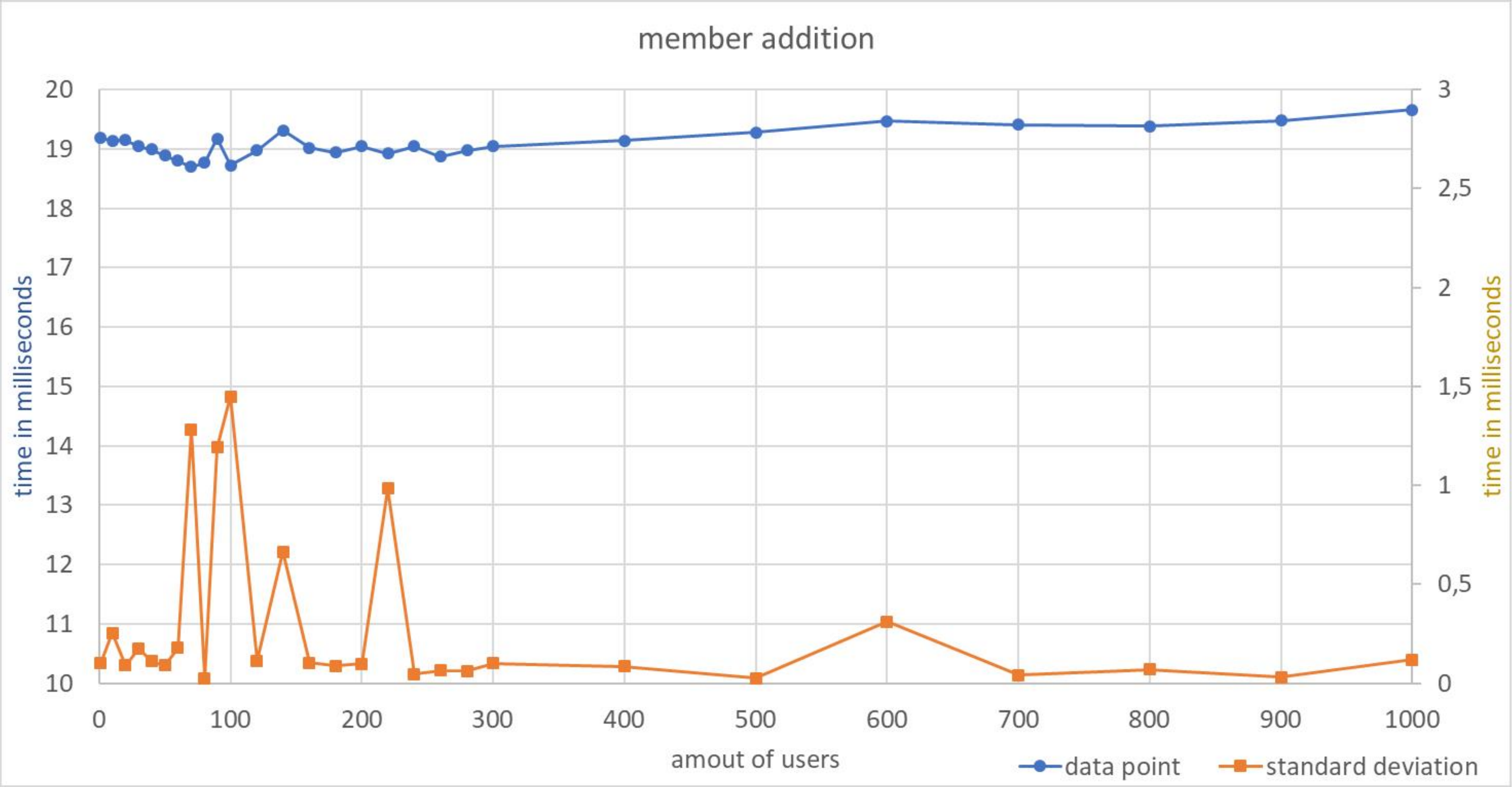


Actions (performance)

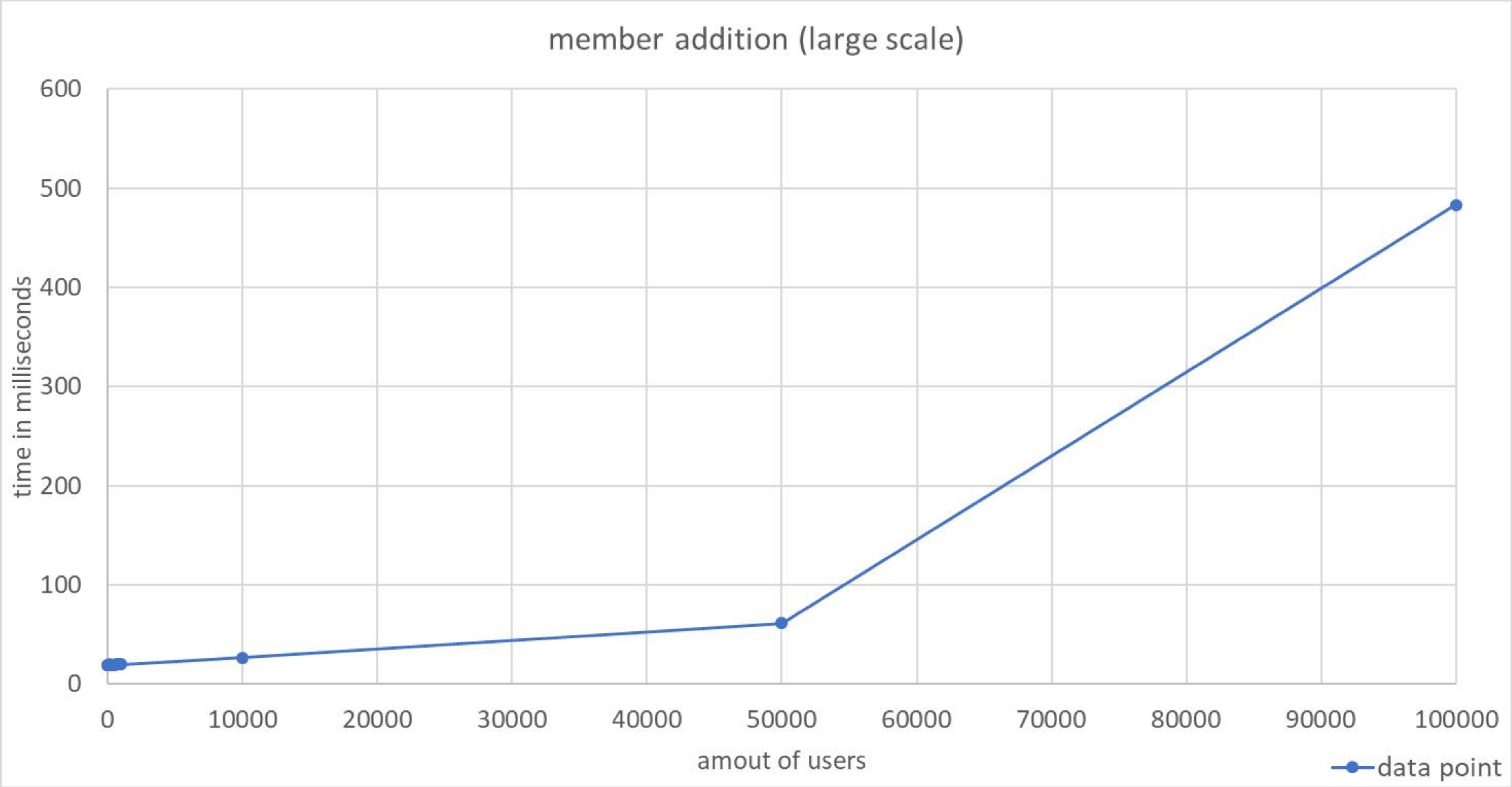
Group changes:



Actions (performance)

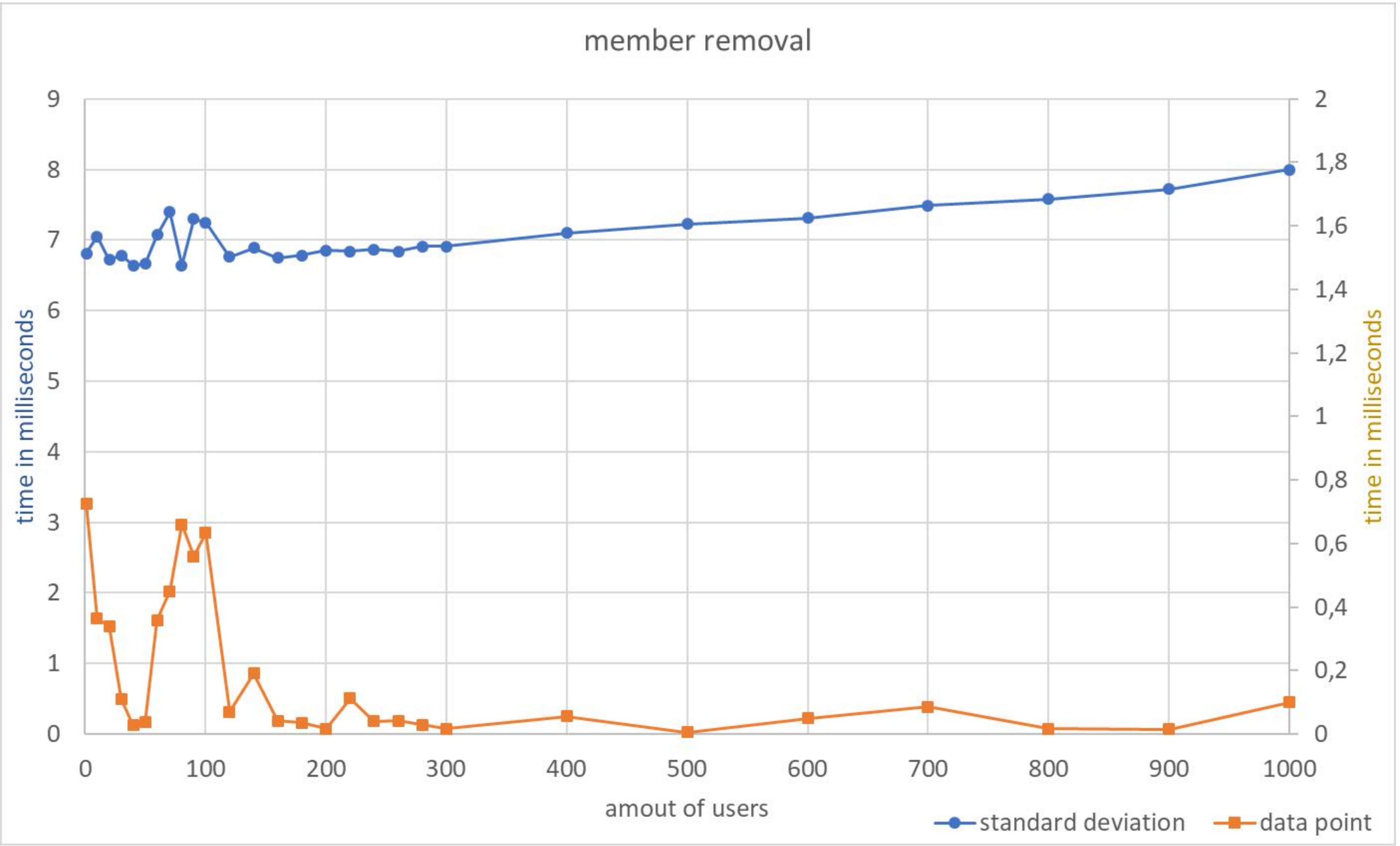


Actions (performance)



Actions (performance)

Problem › Idea › Benefit › **Actions**

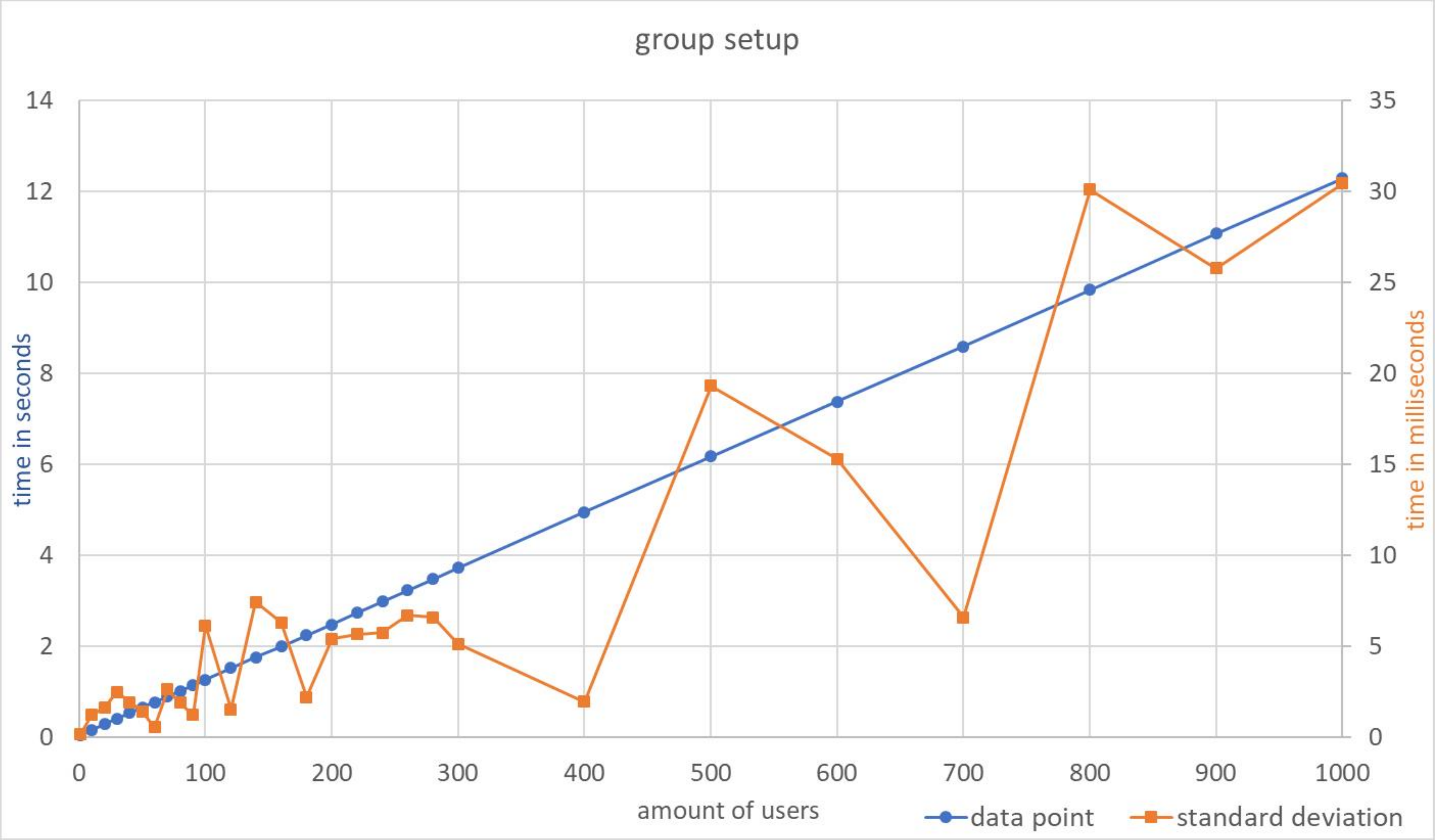


Actions (performance)

Problem › Idea › Benefit › **Actions**

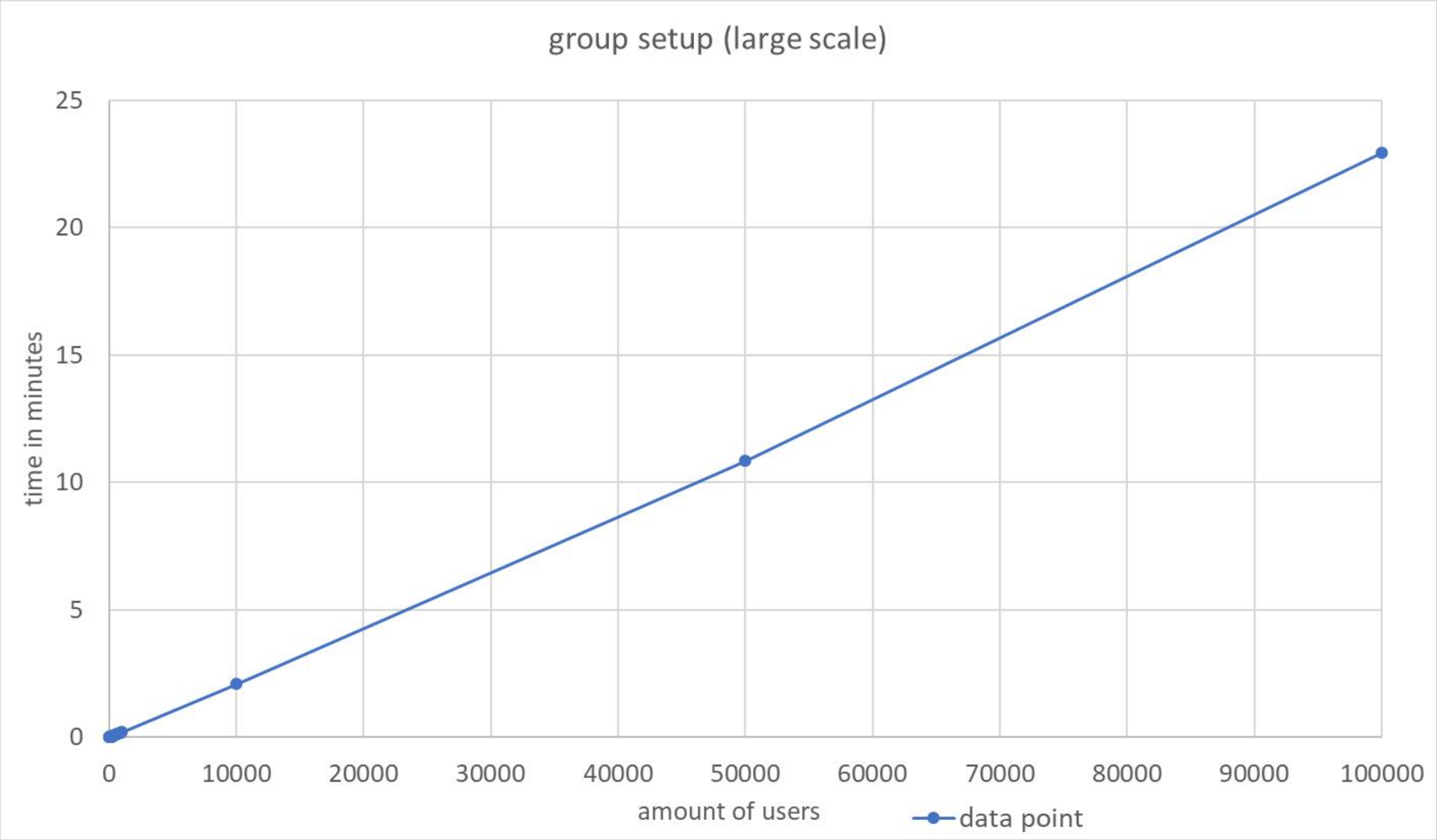
Actions (performance)

Problem › Idea › Benefit › **Actions**



Actions (performance)

Problem ❯ Idea ❯ Benefit ❯ **Actions**



Summary

- **P**roblem: Need for dynamic group-encryption
- **I**dea: Nishat’s dynamic group encryption scheme
- **B**enefit: True dynamic group encryption scheme
- **A**ction: Implementation and evaluation of proposed scheme