Improvements on Distributed Key Generation

Bachelor Project

Kopiga Rasiah

Responsible
Bryan Ford

Supervisor
Nicolas Gailly
Improvements on Distributed Key Generation

- Objective: Bringing improvements in order to enhance the security of the protocol
Outline

• Background:
  • What is DKG
  • Shamir’s secret
  • Feldman’s VSS
• How DKG works
• My work: Proactive secret sharing
• Implementation
• Conclusion
Distributed Key Generation

- Set of $n$ participants who collectively generate a shared private/public key
- Each node have a \textit{share} of the secret (private key)
- No single point failure: attacker needs to break into multiple location to have access to the secret.
- DKG is mostly used in group digital signature, or decrypt shared ciphertexts.
Shamir’s secret sharing
Shamir’s secret sharing
Shamir’s secret sharing

• \( f(x) = s + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}, \quad t < n \)

• \( f(0) = \text{secret} \)

• construct \( n \) points out of it (shares) and distributes to the nodes
Shamir’s secret sharing

- $t$ points are sufficient to reconstruct a $t-1$ degree polynomial function.

1 2 3 4 5
Shamir’s secret sharing
Feldman’s verifiable secret sharing

- Based on Shamir’s secret sharing
- nodes can verify if their shares are consistent
- dealer broadcasts $F(\cdot) = f(\cdot) \cdot g$
- $F(i) == s_i \cdot g$
Distributed Key Generation

• Based on Feldman’s VSS

• System without any trusted party

• Executes $n$ VSS instances in parallel: every node is a dealer

• Each node generates $f_i(x) = z_i + a_1x + a_2x^2 + \ldots + a_{t-1}x^{t-1}$, where $z_i$ is random
Distributed Key Generation

1 \rightarrow f_1(1) \rightarrow 2

1 \rightarrow f_1(3) \rightarrow 3

2 \rightarrow f_1(2) \rightarrow 4

3 \rightarrow f_1(4) \rightarrow 4
Distributed Key Generation

1. $f_1(1)$

2. $f_1(2)$

3. $f_1(3)$

4. $f_1(4)$

1. $f_2(1)$

2. $f_2(2)$

3. $f_2(3)$

4. $f_2(4)$
Distributed Key Generation

\[
f_1(1) + f_2(1) + f_3(1) + f_4(1) = S_1
\]

\[
f_1(2) + f_2(2) + f_3(2) + f_4(2) = S_2
\]

\[
f_1(3) + f_2(3) + f_3(3) + f_4(3) = S_3
\]

\[
f_1(4) + f_2(4) + f_3(4) + f_4(4) = S_4
\]
Distributed Key Generation

\[ f_1(1) + f_2(1) + f_3(1) + f_4(1) = s_1 \]

\[ f_1(2) + f_2(2) + f_3(2) + f_4(2) = s_2 \]

\[ s = \sum_j f_j(0) \]

\[ S = \sum_j F_j(0) = s \times g \]
Proactive secret sharing

- Given enough time, an attacker can gradually break into more than t servers
- Not practical to change the secret
- Solution: Proactive secret sharing.
- We only focus on refreshing the shares
Proactive secret sharing

• Why refreshing?

• Refreshing the shares makes the underlying polynomial change!

• Old stolen information become useless
The idea

• Let’s assume that the initial DKG round has been done

• Each node generates new intermediate random polynomials $g_i(x)$

  $$g_i(x) = 0 + b_{1,i}x + b_{2,i}x^2 + \ldots + b_{t-1,i}x^{t-1}$$

• They execute again the DKG protocol:

• distributions of the intermediate shares
Distributed Key Generation

\[ g_1(x) \]

\[ g_2(x) \]

\[ g_1(1) \]

\[ g_2(1) \]

\[ g_1(3) \]

\[ g_2(3) \]

\[ g_1(2) \]

\[ g_2(2) \]

\[ g_1(4) \]

\[ g_2(4) \]

\[ g_3(x) \]

\[ g_4(x) \]
Proactive secret sharing

\[ s_i = \sum_j f_j(i) \quad \text{for node i} \]

\[ s_i' = \sum_j g_j(i) \]
Proactive secret sharing

\[ s_i = \sum_j f_j(i) \]

\[ + \quad s'_i = \sum_j g_j(i) \leftarrow 2^{nd} \text{round DKG} \]

\[ r_i = \sum_j h_j(i) \]
Proactive secret sharing

\[ s_i = \sum_j f_j(i) \]

\[ + \quad s'_i = \sum_j g_j(i) \]

\[ r_i = \sum_j h_j(i) \]
Proactive secret sharing

\[ s_i = \sum_j f_j(i) \]
\[ + s'_i = \sum_j g_j(i) \]
\[ r_i = \sum_j h_j(i) \]

\[ s = \sum_j f_j(0) \]
\[ + s'_i = \sum_j g_j(0) = 0 \]
\[ s = \sum_j h_j(0) \]

\[ g_i(x) = 0 + b_{1,i}x + b_{2,i}x^2 + \ldots + b_{t-1,i}x^{t-1} \]
Distributed Key Generation

\[
\begin{align*}
    s_1 + \sum_j g_j(1) &= r_1 \\
    s_2 + \sum_j g_j(2) &= r_2 \\
    s_3 + \sum_j g_j(3) &= r_3 \\
    s_4 + \sum_j g_j(4) &= r_4
\end{align*}
\]
Distributed Key Generation

\[
\begin{align*}
 s_1 + \sum_j g_j(1) &= r_1 \\
 s_2 + \sum_j g_j(2) &= r_2 \\
 s_3 + \sum_j g_j(3) &= r_3 \\
 s_4 + \sum_j g_j(4) &= r_4
\end{align*}
\]

renewed share
Proactive secret sharing

- The attacker’s time is now restricted between the updating process
- He need to break into servers at the same period
Implementation

- 2\textsuperscript{nd} round of DKG for updating the shares:

- Renew function adds 2 shares according to their indices:
  
  - check if $G(0) = 0$ ($= 0 \times g$)
  
  - check $\text{share1.index} == \text{share2.index}$
Conclusion

• enhances security of the protocol

• much more interesting if periodicity is implemented

Future work
• Implement the periodicity
• Implement the share recovering process

Current work
• Drand (distributed randomness beacon daemon) where
• nodes collectively produces random values