Project Report **Rachit Rawat**
Prof. Mahavir Jhawar Date: 12/12/18

Project Statement

Study and Implementation of Distributed SSH Key Management with Proactive RSA Threshold Signatures

Abstract

Large enterprises often face difficulty in managing the high number of SSH keys. In this report, we study and implement ESKM - a distributed enterprise SSH key manager [1]. ESKM is a secure and fault-tolerant logically-centralized SSH key manager. In ESKM architecture, SSH private keys are never stored at any single node. Instead, each node only stores a share of the private key. These shares are refreshed at regular intervals for enforced security. For signing, the system uses k-out-of-n threshold RSA signatures.

Notations

- k : threshold value
- ℓ : number of nodes
- n : RSA public key modulus
- e : RSA public exponent
- d : RSA private exponent
- p, q : safe primes
- Δ : $\ell!$
- s_i : current share of node i
- x_i : signature fragment by node i
- H : digest to be signed
- σ : final signature
- sk : RSA private key
- pk : RSA public key
- dk : RSA dummy private key

1 The Security Manager

- KeyGen: Takes b $\in \{1024, 2048, 4096\}$ as input and proceeds as follows:
  - Generate sk and dk b-size RSA key:
    ```
    openssl genrsa -out sk.pem [b]
    ```
  - Get pk from sk:
    ```
    openssl rsa -in sk.pem -outform PEM -pubout -out pk.pem
    ```
  - Convert pk into $\mathsf{pk}_{\mathsf{SSH}}$:
    ```
    ssh-keygen -f pk.pem -i -mPKCS8
    ```
  - Extract e, n, p, q from sk with regex
  - Set $p' = \frac{p-1}{2}, q' = \frac{q-1}{2}$

- Set $m = p'q'$
- Set $d = e^{-1} \bmod m$
- Output $\mathsf{pk}_{\mathsf{SSH}}$, dk

- SplitSecret: Takes d as input and proceeds as follows:
  - Set $a_0 = d$
  - Choose $a_i \in_R \{0, 1, .., m-1\}\ \forall\ 1 \le i \le k-1$.
  - Set $f(x) = \sum_{i=0}^{k-1} a_i x^i$
  - Compute $s_i = f(i) \bmod m, \forall\ 1 \le i \le \ell$
  - Output $s_i, \forall\ 1 \le i \le \ell$

  Note: All group operations from here will be done mod n unless mentioned otherwise.

- GenVerifyInfo: Takes p, q as input and proceeds as follows:
  - Set g_p = Primirive Root of p
  - Set g_q = Primitive Root of q
  - Use the Chinese Remainder Theorem to get the primitive root of n:
    * $\bar{g} = g_p \bmod p$
    * $\bar{g} = g_q \bmod q$
  - Set $g = \bar{g}^2$
  - Output $g^{a_i}, \forall\ 0 \le i \le k-1$

- Broadcast: Proceeds as follows:
  - Send $\mathsf{pk}_{\mathsf{SSH}}$, dk to client
  - $\forall\ 1 \le i \le \ell$, send $(g^{a_j}, \forall 0 \le j \le k-1), s_i$ to node i
  - Delete sk

2 The Control Cluster

- VerifyShare: Takes s_i as input and proceeds as follows:
  - Verify $g^{s(i)} = \prod_{j=0}^{k-1} (g^{a_j})^{i^j}$

- Sign: Takes H as input and proceeds as follows:
  - Compute $x_i = H^{2\Delta s(i)}$
  - Send x_i to client

- RefreshShare: Takes s_i as input and proceeds as follows:
  - Set $a_0 = 0$
  - Choose $a_i \in_R \{0, 1, .., n-1\}\ \forall\ 1 \le i \le k-1$.
  - Set $z(x) = \sum_{i=0}^{k-1} a_i x^i$

- $\forall\ 1 \leq i \leq \ell$, send $z(i)$ to node i
- $\forall\ 1 \leq i \leq \ell$, receive s'_i from node i
- If $|\{s'_1.s'_2, .., s'_\ell\}| \geq k$
  * Compute $s^*_i = s_i + \sum s'_i \in \mathbb{Z}$
  * Set $s_i = s^*_i$
- Repeat RefreshShare after every 60 seconds ‘

3 The Client

- CombineSig: Takes x_i from at least k servers as input and proceeds as follows:
  - Compute $w = \prod_i {x_i}^{2\lambda^s_i}$ where, $\lambda^s_i = \Delta\lambda_i \in \mathbb{Z}$
  - Compute $a = (4\Delta^2)^{-1} \bmod e$
  - Compute $b = \frac{(1-(4\Delta^2 a))}{e} \in \mathbb{Z}$
  - Compute $\sigma = w^a \times H^b$
  - Output σ

4 Algorithms Implemented

- Square and Multiply: exponentiation using pow()
  Note: If $c < 0, x^c \equiv (x^{-1})^{|c|}$
- Extended Euclidean Algorithm: modular multiplicative inverse
- Primitve Root of (safe prime)
  Note: x is a primitive root if $\forall$ prime factors p_i of $(p-1), x^{(p-1)/p_i} \not\equiv 1 \bmod p$
- Chinese Remainder Theorem: primitive root of n
- Horner's Method: polynomial evaluation

5 Miscellaneous

- Patched OpenSSL:
  - Default install location: /usr/local/ssl/bin
  - Generates sk with p and q as safe primes
  - Generates dk with random values for all fields except for e and n.
- Working directory: /tmp
- All communications are over TLS

6 Source Code

- Core
- OpenSSL

References

[1] Y. Harchol, I. Abraham, and B. Pinkas. Distributed ssh key management with proactive rsa threshold signatures. Cryptology ePrint Archive, Report 2018/389, 2018. https://eprint.iacr.org/2018/389

[2] Tatu Ylonen: SSH - Secure Login Connections over the Internet. Proceedings of the 6th USENIX Security Symposium, pp. 37-42, USENIX, 1996.