Secure and Efficient Key Derivation in Portfolio Authentication Schemes Using Blakley Secret Sharing

Peter Mayer, Melanie Volkamer
Portfolio authentication

![Diagram of portfolio authentication process]

- e1, e2, e3, e4, e5, e6, e7: Elements of the password.
- c1, c2, c3, c4: Challenge set.
- e1', e2', e3', e4': User input.
- Stored Information
- Password Verification
- ...
Why portfolio authentication?

<shoulder surfing image>
Graphical recognition-based portfolio authentication
Graphical recognition-based portfolio authentication
Portfolio authentication

- Naïve approach
  - Generate & store the hashes for all authorized subsets
  - Check if user input matches one of the hashes

- Easy to implement

- Factorial growth of number of hashes
  \[ \rightarrow \text{inefficient} \]
(t,n)-threshold verification

challenge response pairs correspond to parties

→ $n = |\text{password}|$

→ $t = |\text{authorized subsets}|$
Secret sharing

Usual Secret sharing

1. Choose construct
2. Choose common secret
3. Generate and distribute shares
4. Combine shares to reconstruct common secret

Shamir secret sharing

1. Dealer chooses polynomial $a$ of degree $t - 1$
2. the common secret is $a_0$
3. Choose shares $y_i = a(x_i)$
4. Reconstruct $a_0$ using Lagrange interpolation
Secret sharing

Usual Secret sharing

1. Choose construct
2. Choose common secret
3. Generate and distribute shares
4. Combine shares to reconstruct common secret

Shamir secret sharing

1. Dealer chooses polynomial $a(x)$ of degree $t - 1$
2. $a_0$ is chosen as the common secret
3. Choose shares $y_i = a(x_i)$
4. Reconstruct $a_0$ using Lagrange interpolation

Problem: in the verification scenario the shares are already predetermined by the password
Blakley secret sharing

- Uses hyperplane geometry in $t$-dimensional space

1. Common secret is the first coordinate of random $t$-dimensional point $x$ in $GF(p)$
2. The dealer chooses $t$ values $m_{ij}$ at random for each party...
3. ...and calculates the shares $y_i$ for all parties $m_{i1}x_1 + m_{i2}x_2 + \cdots + m_{it}x_t = y_i$
4. To reconstruct the secret, $t$ parties combine their shares to solve $M'x = y'$
Blakley secret sharing

- As is, the problem persists
- but it is possible to adapt Blakley secret sharing allowing preselection of the shares

1. Common secret is the first coordinate of random $t$-dimensional point $x$ in $GF(p)$
2. The dealer chooses $t$ values $m_{ij}$ at random for each party...
3. ...and calculates the shares $y_i$ for all parties $m_{i1} + m_{i2} + \cdots + m_{it} = y_i$
4. To reconstruct the secret, $t$ parties combine their shares to solve $M'x = y$
(t,n)-threshold verification

Procedure

Enrollment

1. Choose prime $p$ and random $t$-dimensional point $x$ in $GF(p)$

2. Derive share $y_i = KDF(e_i)$ from password elements $e_i$

3. Choose coefficients $m_{i1}, ..., m_{it-1}$

4. Solve $m_{i1} + m_{i2} + ... + m_{it} = y_i$ for $m_{it}$

5. Store $M$ and $s$
(t,n)-threshold verification

Procedure

Verification
1. Gather user input $e'_i$
2. Derive shares $y'_i$ from the user input $e'_i$
3. Build $M$ from the $m_{ij}$ corresponding to the challenges
4. Solve $M'x = y'$ for $x$
5. Check whether $KDF(x_1) \stackrel{?}{=} s$

$$(m'_{11} \ldots m'_{1t}) (x_1 \ldots x_t) = (y'_1 \ldots y'_n)$$

$s \stackrel{?}{=} KDF(x_1)$
Security properties

**Guessing Resistance**
- All variables in $GF(p)$
- $p$ distinct values
  - on average $\frac{p}{2}$ attempts
- Perfect secret sharing
- To ensure strength $H$:

\[ p \geq 2^{H+1} \]

**Secure Storage**
- Security of storage completely depends on used KDF
- If KDF is secure, $(t,n)$-threshold verification is secure
Efficiency properties

- Evaluation against naïve approach
  - Only alternative available

- Storage

- Computation time
Storage comparison

Two security levels

- PIN-level (password space of $10^4$ entries)
  - Example: 4 out of 6, alphabet: 0-9

- Password-level (password space of $\sim 2^{27}$ entries)
  - Example: 6 out of 9, alphabet: 95 keys on US standard keyboard
Storage comparison

<table>
<thead>
<tr>
<th>PIN-level</th>
<th>Password-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive approach</td>
<td>(t,n)-threshold verification</td>
</tr>
<tr>
<td>Storage requirement (bytes)</td>
<td></td>
</tr>
<tr>
<td>480</td>
<td>2688</td>
</tr>
<tr>
<td>80</td>
<td>248</td>
</tr>
</tbody>
</table>

6 times higher storage requirement in naïve approach

10 times higher storage requirement in naïve approach
Computation time

- Monte Carlo evaluation
  - 10,000 iterations
- Implementation in Mathematica
  - Salted SHA-256 as KDF
  - Code available on GitHub
    - https://github.com/SecUSo/t-n-threshold-verification
Computation time comparison

All timings are smaller than the usual 10,000 iterations of SHA-256 recommended for password storage on the same platform.
Summary

- $(t,n)$-threshold verification
  - Enabler for new authentication technologies
  - More efficient in terms of storage and time
  - Allows secure & efficient storage in portfolio authentication schemes for the first time
  - Allows additional operations to easily modify the password
What’s next?

- Portfolio authentication originally conceived for graphical recognition-based schemes

- Applications go far beyond this limited scenario
(t,n)-threshold verification

- Allows arbitrary choice of passwords
- Enables use of graphical recognition-based passwords and other more diverse setups
- Creation faster than naïve approaches in relevant scenarios
- Supports additional operations in comparison to naïve approach
- 6 to 10 times less storage than naïve approach
- Verification faster for higher numbers of authorized subsets

Supports additional operations in comparison to naïve approach