Password-Based Cryptography: Strong Security from Weak Secrets

Anja Lehmann
IBM Research – Zurich
based on joint work with Jan Camenisch, Anna Lysyanskaya & Gregory Neven
ROADMAP

- **Password-Based Authentication**
  How to make password checking systems *even* better

- **Password-Authenticated Secret Sharing**
  How to make cryptography accessible to end users
Password-Based Authentication

- Most prominent form of user authentication – convenient! No key, software, ...

Password rules:
- upper and lower case letters and numbers at least 16 characters in length
- never reuse your password on another site
- change your passwords periodically

stores only (salted) password hashes $h = \text{Hash}(\text{pwd})$

vs. 4-digit PIN for ATM cards
- why the difference?

the ATM will retain the card after 3 failed attempts!

<table>
<thead>
<tr>
<th>Username</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>wb3822Ujsd4</td>
</tr>
<tr>
<td>Bob</td>
<td>b5kMsa8dsbn</td>
</tr>
<tr>
<td>Carol</td>
<td>77peCu52Kry</td>
</tr>
</tbody>
</table>

$h' = h$ ?
Password-Based Authentication

- If service provider is trusted & throttles after too many failed attempts → short passwords are sufficient!

- **But** main threat to password security is server compromise

- The more complicated our passwords are, the more guesses the adversary need

  NIST: 16-character passwords have 30 bits of entropy ~ 1 billion possibilities

  vs.

  $150$ GPUs can test ~ 300 billions/second
Passwords inherently insecure?

No! We’re just using them incorrectly ...
Password-Based Authentication Done Right

- Offline attacks are inherent in single-server setting
- Solution: split password verification over multiple servers

<table>
<thead>
<tr>
<th>Username</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>wb3822Ujsd4</td>
</tr>
<tr>
<td>Bob</td>
<td>b5kMsa8dsbn</td>
</tr>
<tr>
<td>Carol</td>
<td>77peCu52Kry</td>
</tr>
</tbody>
</table>

Service Provider

Password correct?

Backend Server 1

Backend Server 2

Backend Server n
**Pythia: OPRF Service**

- Replace *Hash* by a secure $\text{PRF}(\text{key}, \text{pwd})$.
- Store at remote server & evaluate PRF obliviously.

---

### Username, Hash

<table>
<thead>
<tr>
<th>Username</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>wb3822Ujsd4</td>
</tr>
<tr>
<td>Bob</td>
<td>b5kMsa8dsbn</td>
</tr>
<tr>
<td>Carol</td>
<td>77peCu52Kry</td>
</tr>
</tbody>
</table>

---

Distributed Password Verification | High-Level Idea

- Replace $Hash$ by a secure $PRF(\cdot, pwd)$
- Split secret key $\cdot$ into $n$ shares
- $h = PRF(\cdot, pwd)$ computed distributed:
  - Servers don’t learn anything about $pwd$ or $h$

### Jointly compute $PRF(\cdot, pwd)$

#### Username, $pwd'$

<table>
<thead>
<tr>
<th>Username</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>wb3822Ujsd4</td>
</tr>
<tr>
<td>Bob</td>
<td>b5kMsa8dsbn</td>
</tr>
<tr>
<td>Carol</td>
<td>77peCu52Kry</td>
</tr>
</tbody>
</table>

Distributed Password Verification | Security

- Secret key has high-entropy, i.e., cannot be guessed
  - Adversary needs backend servers (or full key) to verify password guesses
  - Backend servers will stop verification if activity is suspicious

<table>
<thead>
<tr>
<th>Username</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>wb3822Ujsd4</td>
</tr>
<tr>
<td>Bob</td>
<td>b5kMsa8dsbn</td>
</tr>
<tr>
<td>Carol</td>
<td>77peCu52Kry</td>
</tr>
</tbody>
</table>

Jointly compute $\text{PRF}(\text{key}, pwd)$
Distributed Password Verification | Proactive Security

- Secret key gets re-shared periodically
  - All previous key shares get useless
  - Adversary must break into all servers at the same time

- As long as one server is not corrupted
  - Passwords are secure

<table>
<thead>
<tr>
<th>Username</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>wb3822Ujsd4</td>
</tr>
<tr>
<td>Bob</td>
<td>b5kMsa8dsbn</td>
</tr>
<tr>
<td>Carol</td>
<td>77peCu52Kry</td>
</tr>
</tbody>
</table>
DPV Protocol

*Optimal Distributed Password Verification. ACM CCS’15.
Camenisch, Lehmann, Neven.*
Distributed Password Verification | Protocol

- Replace *Hash* by a secure $H(uid, pwd)^k$
- Split secret key $k = k_1 + k_2 + \ldots + k_n \mod q$

$k = \text{random element in } \mathbb{Z}_q$

Cyclic group of prime order $q$

Naor, Pinkas, Reingold. *Distributed Pseudorandom Functions and KDCs*. Eurocrypt '99
- Replace \textit{Hash} by a secure $H(uid, pwd)^k$.
- Split secret key $k = k_1 + k_2 + \ldots + k_n \mod q$.

Naor, Pinkas, Reingold. \textit{Distributed Pseudorandom Functions and KDCs}. Eurocrypt '99
Distributed Password Verification | Protocol

- Replace $\text{Hash}$ by a secure $H(uid, pwd)^k$
- Split secret key $k = k_1 + k_2 + \ldots + k_n \mod q$

Service Provider

$U = H(uid, pwd)^N$

$V = \prod V_i^{1/N} = U^{k_1 + k_2 + \ldots + k_n}$

$h = H'(uid, pwd, V)$
Distributed Password Verification | Protocol

- Replace \textit{Hash} by a secure $H(uid, pwd)^k$
- Split secret key $k = k_1 + k_2 + ... + k_n \mod q$

\[ V_n = U^{k_n} \]

\[ V_1 = U^{k_1} \]

\[ V_2 = U^{k_2} \]

+ blinding for adaptive security

+ pairing for correctness check (only at setup)

\[ h = H'(uid, pwd, V) \]
Distributed Password Verification | Protocol

- Proactive security & re-sharing of keys:

\[ k = k'_1 + k'_2 + \ldots + k'_n \mod q \]

Agree on pseudorandom shares of zero:

\[ \delta_1 + \delta_2 + \ldots + \delta_n = 0 \mod q \]

- No updates of “hash table” needed!

+ non-interactive protocol for computing \( \delta_i \) (leveraging trusted setup & “secure” backup)
Distributed Password Verification
= Distributed OPRF (Oblivious PRF)

compute $y = \text{PRF}(k, x)$ in a blind & distributed manner
Distributed Password Verification
= Distributed OPRF (Oblivious PRF)

compute $y = \text{PRF}(k, x)$ in a blind & distributed manner

$k = \text{KGen}(\tau)$
$k_1 + k_2 + \ldots + k_n = \text{Share}(k, n)$

$x = \text{Blind}(x)$

$\bar{y} = \text{Comb}(\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_n)$

$y = \text{Unblind}(\bar{y})$

s. t. $y = \text{PRF}(k, x)$
Distributed Password Verification | Security & Efficiency

- Efficient & round-optimal protocol
  - 1 round of communication
  - Login: one exponentiation per server (two for SP)
  - Non-interactive key refresh
  - Prototype implementation & evaluation (Ergon)
    - 3 backend servers, each 16 x 2.9Ghz core: 285 logins/second

- Provable security in very strong security model
  - Adaptive & active adversaries, UC Framework
  - One-More Gap DH (OMGDH), Random Oracle

- Password protection back where it belongs: on the server!
ROADMAP

- **Password-Based Authentication**
  How to make password checking systems *even* better

- **Password-Authenticated Secret Sharing**
  How to make cryptography accessible to end users
How to bridge cryptographic keys & humans

- Most cryptography relies on strong secret keys
- Easy to manage for servers and devices ... not so easy for humans

---BEGIN PRIVATE KEY---
MIICXgIBAAKBgQDHikastc8+I81zCg/qWW8dMr8mqvXQ3qBPAmudRjxoZVI47vts
kyIFAXOFsPh0O2nUuooJngnHV0639jTTEYG1VckNaW2R6USQ7dSf5q5u+uV3pMk
7wVs4n3urG6jntq7rXbcC1bNa/PFeA2abf7TfFBBy0IG00zc128ishywcIDAQAQ
AoGBALTNi2zJvxy4qSxW3VH01b0eM6b2iqb5gJiWb11FgaOD77nGkFWc
bYgfpS5qrq4yhvLAXxv8bcsqxORIfHyV68U1y4R+8WxDNh0aevex8HR5/3X5
031D1mJ1UOE+StltXNc3ebHShE+10x0bHSZ+WOQLY71JakEAAsjRgKp85cG
auUjvdOl2zHllWQY5Wz+nOHUuw8P7wOTKU3AOJAEWeE771p9p/GTA/krdBQnPP
QvWUJDxGjwBAND5C6krwPewFyKrKtjOGJ1ImioY72wrRnoNcdEES3hDROh/f48YWFo
rLbZy1zzZNFy/gm2X6XJQftGq+ZD9UCQGljaHxGmpDUaHtxsGsUsYTr
irOx0Dlq7zhE6935aBG010DF6MLzIoZ1Cm55RtuVAcSLZEL/215UCCQQA
a3MXu4h91L9HMY1W7PDQsaORcgkY77bSNGk+u9WS/bH1uYlAIPSFUZ
btt+j1c1UigSw+aeeZljAKExurpmpqj5vG087ZngKFFGR5rozDI7sK5dceTV97K
a3Y+Nzi+XWxTxB8WkYmzKv402h2EFWBYuDnS5ZH/bw==
-----END PRIVATE KEY-----

E.g., encrypted cloud storage (untrusted cloud)

How to store the secret key?
- Access from many devices
- Trusted hardware inconvenient
- Device(s) can get broken or lost
Secret Sharing | Shamir’ 79

user shares secret $K$ with $n$ servers

How to ensure it's the legitimate user?

user retrieves $K$ from at least $t+1$ servers

$t+1$ shares needed to reconstruct $K$
if at most $t$ servers are corrupt → they don't learn anything about $K$
Password-Authenticated Secret Sharing | BJSL’11

user shares secret $K$ with $n$ servers protected by password $p$

user retrieves $K$ from at least $t+1$ servers using password $p'$

$t+1$ shares needed to reconstruct $K$ and to verify whether $p = p'$
if at most $t$ servers are corrupt → they don't learn anything about $K$ or can offline attack $p$
honest server throttle verification after too many (failed) attempts

Password-Authenticated Secret Sharing (TPASS/PPSS)

User shares secret $K$ with $n$ servers protected by password $p$

User retrieves $K$ from at least $t+1$ servers using password $p'$

User has to remember the servers she trusted at setup
Password-Authenticated Secret Sharing (TPASS/PPSS)

user shares secret $K$ with $n$ servers protected by password $p$

user retrieves $K$ from at least $t+1$ servers using password $p'$

if user gets tricked into retrieval with $t+1$ corrupt servers → password $p'$ is leaked

## Overview of TPASS Solutions

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Security Model</th>
<th>Assumption</th>
<th>Rounds</th>
<th>Retrieval Exponentiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BJSL’11</td>
<td>Game</td>
<td>DDH-ROM</td>
<td>3</td>
<td>8t+17 16</td>
</tr>
<tr>
<td>CLLN’14</td>
<td>UC</td>
<td>DDH-ROM</td>
<td>5</td>
<td>14t+24 7t+28</td>
</tr>
<tr>
<td>JKK’14</td>
<td>Game</td>
<td>OMGDH-ROM</td>
<td>1</td>
<td>2t+3 3</td>
</tr>
<tr>
<td>ACNP’16</td>
<td>Game</td>
<td>OMGDH-ROM</td>
<td>1</td>
<td>? ?</td>
</tr>
<tr>
<td>JKKX’16</td>
<td>UC</td>
<td>OMGDH-ROM</td>
<td>1</td>
<td>t+2 1</td>
</tr>
<tr>
<td>JKKX’17</td>
<td>UC</td>
<td>TOMGDH-ROM</td>
<td>1</td>
<td>2 1</td>
</tr>
</tbody>
</table>
SECURITY MODELS
for password-based crypto
**Provable Security**

- Old days: security by obscurity

- Now: provable security = gold standard in cryptography
  - Formal security model $\&$ formal security proof
  - Also crucial for higher-level protocols: secure building blocks $\Rightarrow$ secure protocol

**Game-Based**

\[ \text{Oracle} \]

\[ \Downarrow \text{attack} \]

**(UC) Ideal vs Real**

- $F$
- Server 1
- Server $n$
Challenge: Security Model including the User

- Game-based security notions most common
  - Oracle access to some secret key function
  - Secure if ∀Adv: Prob[attack] = negligible

- User/Password-based cryptography
  - Adversary has black-box access “to the user”

<table>
<thead>
<tr>
<th>Model</th>
<th>Reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passwords chosen at random from known, independent distribution</td>
<td>People reuse passwords, leak info about passwords</td>
</tr>
<tr>
<td>Honest user always uses correct password</td>
<td>Users make typos, “mix” passwords</td>
</tr>
</tbody>
</table>
Security defined via ideal functionality $\mathcal{F} - \mathcal{F}$ is “secure-by-design”
Security defined via ideal functionality $\mathcal{F}$ – $\mathcal{F}$ is “secure-by-design”
Universal Composability Framework | Canetti’00

- Security defined via ideal functionality $\mathcal{F} - \mathcal{F}$ is “secure-by-design”
- Protocol $\pi$ securely implements $\mathcal{F}$ if $\forall \text{Adv} \exists \text{Sim}$ such that $\forall E: \text{REAL}_{\pi,A,E} \approx \text{IDEAL}_{\mathcal{F},S,E}$.

\[
\text{environment chooses passwords of honest users} \Rightarrow \text{no assumptions on pwd distributions \& typos by honest users covered}
\]
## Overview of TPASS Solutions

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Security Model</th>
<th>Assumption</th>
<th>Rounds</th>
<th>Exponentiation User</th>
<th>Exponentiation Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>BJS’11</td>
<td>Game</td>
<td>DDH-ROM</td>
<td>3</td>
<td>$8t + 17$</td>
<td>16</td>
</tr>
<tr>
<td>CLLN’14</td>
<td>UC</td>
<td>DDH-ROM</td>
<td>5</td>
<td>$14t + 24$</td>
<td>$7t + 28$</td>
</tr>
<tr>
<td>JKK’14</td>
<td>Game</td>
<td>OMGDH-ROM</td>
<td>1</td>
<td>$2t + 3$</td>
<td>3</td>
</tr>
<tr>
<td>ACNP’16</td>
<td>Game</td>
<td>OMGDH-ROM</td>
<td>1</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>JKKX’16</td>
<td>UC</td>
<td>OMGDH-ROM</td>
<td>1</td>
<td>$t + 2$</td>
<td>1</td>
</tr>
<tr>
<td>JKKX’17</td>
<td>UC</td>
<td>TOMGDH-ROM</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Disclaimer: security models vary

All based on OPRFs
TPASS by JKKX’17 | Slightly Different Setting

user shares secret $K$ with $n$ servers protected by password $p$

user retrieves $K$ from at least $t+1$ servers using password $p'$

user obtains a random key $K$ at setup

if $< t+1$ servers are corrupt → they don't learn anything about $K$

if $\geq t+1$ servers are corrupt → they learn $K$ (but its still a random key)

[JKKX’17] Jarecki, Kiayias, Krawczyk, Xu. TOPPSS: Cost-Minimal Password-Protected Secret Sharing Based on Threshold OPRF. ACNS 2017
Building Block: Threshold OPRF (T-OPRF)

compute $y = \text{PRF}(k, x)$ in a blind & distributed threshold manner

$k = \text{KGen}(\tau)$

$k_1 + k_2 + \ldots + k_n = \text{Share}(k)$

any $t + 1$ shares are sufficient to compute PRF

$x = \text{Blind}(x)$

$y = \text{Unblind}(\bar{y})$

s.t. $y = \text{PRF}(k, x)$

If $< t+1$ servers are corrupt:

T-OPRF outputs are indistinguishable from random

can only evaluate PRF with help of honest servers
TPASS Protocol | Setup

- user obtains secret $K$ protected by password $p$ with $n$ servers $SS = S_1, S_2, ..., S_n$

\[ k = \text{PRF}(\text{KGen}(\tau)) \]
\[ (k_1, k_2, ..., k_n) = \text{Share}(k, t, n) \]

if ack from all $S$ in $SS$
- compute $y = \text{PRF}(k, p)$
- compute $h = H(y)$
- parse $h = (C, K)$
TPASS Protocol | Setup

- user obtains secret $K$ protected by password $p$ with $n$ servers $SS = S_1, S_2, ..., S_n$

\[ k = \text{PRF.} \text{KGen}(\tau) \]
\[ (k_1, k_2, ..., k_n) = \text{Share}(k, t, n) \]

if \text{ack} from all $S$ in $SS$
- compute $y = \text{PRF}(k, p)$
- compute $h = \text{H}(y)$
- parse $h = (C, K)$

send $C$ to all $S$ & output $K$
TPASS Protocol | Setup

- user obtains secret $K$ protected by password $p$ with $n$ servers $SS = S_1, S_2, ..., S_n$

\[ k = \text{PRF} \cdot \text{KGen}(\tau) \]
\[ (k_1, k_2, ..., k_n) = \text{Share}(k, t, n) \]

if ack from all $S$ in $SS$
- compute $y = \text{PRF}(k, p)$
- compute $h = H(y)$
- parse $h = (C, K)$

send $C$ to all $S$ & output $K$

$K$ is always a random key
If $< t+1$ servers are corrupt: Adv learns nothing about $(p, K)$
If $\geq t+1$ servers are corrupt: Adv can offline attack $(p, K)$
TPASS Protocol | Retrieval

- user retrieve her secret using password $p'$ from $t+1$ servers $SR = S'_1, S'_2,...,S'_{t+1}$

$\bar{x} = \text{Blind}(p')$

Each $S_i$: check that $SR \subset SS$
compute $\overline{y}_i = \text{pPRF}(k_i, \bar{x})$
TPASS Protocol | Retrieval

- User retrieve her secret using password \textit{pwd}' from \(t+1\) servers \(SR = S'_1, S'_2, ..., S'_{t+1}\)

\[\text{uid, } p', \text{ } SR\]

\(\bar{x} = \text{Blind}(p')\)

If \((C, \bar{y}_i)\) from all \(S\) in \(SR\)
compute \(\bar{y} = \text{Comb}(\bar{y}_1, \bar{y}_2, ..., \bar{y}_{t+1})\)
compute \(y = \text{Unblind}(\bar{y})\)
compute \(h = H(y)\)
parse \(h = (C', K')\)

If \(C' = C\) output \(K'\)
else output \(K' = \perp\)

Security based on T-OPRF & ROM

Efficient T-OPRF from OMGDH & ROM (similar to our DORPF)

each \(S_i\): check that \(SR \subseteq SS\)
compute \(\bar{y}_i = \text{pPRF}(k_i, \bar{x})\)
TPASS Protocol | Retrieval

- user retrieve her secret using password *pwd’* from *t+1* servers \( SR = S'_1, S'_2, ..., S'_{t+1} \)

\[ \text{uid, p', SR} \]

\[ \bar{x} = \text{Blind}(p') \]

if \((C, \bar{y}_i)\) from all \(S\) in \(SR\) compute \(\bar{y} = \text{Comb}(\bar{y}_1, \bar{y}_2, ..., \bar{y}_{t+1})\)
compute \(y = \text{Unblind}(\bar{y})\)
compute \(h = H(y)\)
parse \(h = (C', K')\)
if \(C' = C\) output \(K'\)
else output \(K' = \bot\)


each \(S_i\):
check that \(SR \subset SS\)
compute \(\bar{y}_i = pPRF(k_i, \bar{x})\)
TPASS | Applications

- TPASS allows users to reconstruct strong secret key from weak password
  - Does not require trusted storage
  - Allows to bootstrap any cryptographic operation based on a strong key
    - Encrypted cloud storage, strong authentication, ...
    - Bootstrap strong “passwords” from K, $pwd = H(K, "iacr.org")$
  - Reconstruction of secret key can be security risk – malware on device

- Less flexible, but more secure: protocols for joint password-based computations
  - Number of “solutions”, most are vulnerable against offline attacks 😞
  - Distributed signing [CLNS16] – “Virtual Smartcard”
Password-Based Crypto | Summary

- Passwords are convenient & easy to use
- Low entropy makes them vulnerable to offline attacks
- Strong security from passwords requires **multi-server solutions**
  - Prevents offline attacks & detect online attacks

- UC-based definitions capture password use better than game-based models
- Highly-efficient solutions exist for a number of password-based primitives
- Lots of open research problems – Lets make crypto for people! 😊

**Thanks! Questions?**
anj@zurich.ibm.com