Security of Internet-Scale Services

Thesis Defense — Adam Everspaugh

Committee:
Prof Nigel Boston
Prof Barton Miller
Prof Somesh Jha
Assc Prof Thomas Ristenpart*
Prof Michael Swift*
Software Environment Has Changed

Microsoft Word, ~1995

- users: $2^0 - 2^5$
- machines: $2^0$

Google Docs, 2015

- users: $2^{20} - 2^{30}$
- machines: $2^{10}$

Diagram showing the evolution from Microsoft Word to Google Docs, highlighting changes in user and machine numbers.
Interesting Properties of Internet-scale Services

- Millions or billions of users
- Geo-replicated applications and storage systems
- Applications built as distributed services: componentized, communication, failures, concurrency
- Highly available: $1.0 - \varepsilon$
- Security?
  - Carried forward from previous era of application development
Research Question

Can we improve the security of internet-scale services?

- **Not-So-Random Numbers [IEEE S&P '14].** Evaluate RNGs in virtual machine and cloud compute environments.

- **Pythia PRF Service [Usenix Sec '15].** Design and evaluate a secure password authentication service built around a new cryptographic primitive.

- **Key Rotation for Auth Encryption [Crypto '17].** Examines updatable encryption for cloud storage. Formal analysis of security notions and updatable encryption schemes.
Random Number Generators

Example uses:

- StackProtector canaries
- TCP/IP sequence numbers
- Cryptographic keys
Random Number Generators

Inputs
system events

RNG

Outputs
uniformly distributed
Random Number Generators

Inputs: system events

RNG

Outputs: uniformly distributed

Linux /dev/(u)random:

Interrupt Pool

Interrupt events

Input Pool

disk events
keyboard events
mouse events
hardware RNGs

Random Pool

/dev/random

URandom Pool

/dev/urandom

get_random_bytes()

Cryptographic hash
Random Number Generators

Inputs → **RNG** → Outputs

- system events
- uniformly distributed

Folklore concerns regarding security

1. Do full-memory snapshots cause problems for system RNGs? [GR05] [RY10]

2. Are input sources entropy-poor inside a virtual machine? [SBW09]
Virtual Machine Snapshot and Resumption

Does the RNG produce distinct outputs with each resumption?
Linux RNG *Not* Reset Secure

**One experiments:**
- Boot VM in Xen, idle for 5 minutes
- Start measurement process, capture snapshot
- Resume from snapshot, read 512-bits from /dev/urandom every 500 us

Repeat for 8 distinct snapshots
Do 20 resumptions/snapshot

7/8 snapshots produce repeated outputs
Why does this happen?

Buffering and thresholds prevent new inputs from impacting outputs.

Linux /dev/(u)random
Reset Vulnerabilities Effect
Other Platforms

FreeBSD
/dev/random produces **identical** output stream
Up to 100 seconds after resumption

Microsoft Windows 7
Produces **repeated** outputs indefinitely
rand_s (stdlib)
CryptGenRandom (Win32)
RngCryptoServices (.NET)
RNG Summary

• Snapshots cause problems? → Yes
• Entropy-poor inputs? → No
• New clean-slate RNG design → Whirlwind
Outline


- Pythia PRF Service [Usenix Sec '15]. Design and evaluate a secure password authentication service built around a new cryptographic primitive.

- Key Rotation for Auth Encryption [Crypto '15]. Examines updatable encryption for cloud storage. Formal analysis of security notions and updatable encryption schemes.
Password Database Compromises

Password DB breaches are common
Website stores one of:

- pw
- Hash(pw)
- salt, Hash(salt, pw)
- salt, Hash^{4096}(salt, pw)

6.5M hashes leaked

90% recovered 2 weeks
Facebook’s Password Onion

```php
$cur  = 'password'
$cur  = md5($cur)
$salt = randbytes(20)
$cur  = hmac_sha1($cur, $salt)
$cur  = remote_hmac_sha256($cur, $secret)
$cur  = scrypt($cur, $salt)
$cur  = hmac_sha256($cur, $salt)
```

[Moffet RWC15]
Facebook’s Password Onion

```php
$cur  = 'password'
$cur  = md5($cur)
$salt = randbytes(20)
$cur  = hmac_sha1($cur, $salt)
$cur  = remote_hmac_sha256($cur, $secret)
$cur  = scrypt($cur, $salt)
$cur  = hmac_sha256($cur, $salt)
```

[Moffet RWC15]
Remote HMAC Distributes Trust

Web Server

pw db

Crypto Server

$secret

hmac($cur, $secret)

How do we rotate $secret?

Hard to detect online attacks
Advantages of Partially Oblivious PRF

- Doesn’t learn secret key
- Detect online attacks
- Doesn’t learn pw

Web Server

```
pw
```

Pythia PRF Service

```
uid, blind(pw)
```

```
y = F_k(uid, blind(pw))
```

```
unblind(y) = F_k(uid, pw)
```

Detect online attacks

Doesn’t learn pw
Existing Crypto Primitives are Insufficient

- Deterministic
- Pseudorandom
- Key Rotation
- PRFs
- Key Updatable Encryption
- Proxy Re-encryption
- Partially-Blind Signatures
- Oblivious PRFs

(Partially Oblivious PRF)
Fast, Scalable PRF Service

**Pythia Query**
- 5.2 ms

**Iterated Hashing**
- 8.9 ms
  - (SHA256$^{10k}$)

**Throughput:**
- 1350 queries/sec
  - (8-core EC2 instance)
- Within factor of 2 of HTTP GET over TLS

**Storage:**
- O(1) per web server
- Supports arbitrary number of users for each web server

**100M Web Server:**
- 18.6 GB (keytable)

Tech Stack:
- nginx
- Django
- MongoDB
- Relic Toolkit
Outline

- **Not-So-Random Numbers [IEEE S&P '14].** Evaluate RNGs in virtual machine and cloud compute environments.

- **Pythia PRF Service [Usenix Sec '15].** Design and evaluate a secure password authentication service built around a new cryptographic primitive.

- **Key Rotation for Auth Encryption [Crypto '17].** Examines updatable encryption for cloud storage. Formal analysis of security notions and updatable encryption schemes.
Encryption for Cloud Storage

$k_1$ — secret key

$\text{file}_0$ $\{\text{file}_0\}_{k_1}$

How do we rotate $k_1$?

$k_1 \rightarrow \Delta_{1 \rightarrow 2}$ $\Delta_{1 \rightarrow 2}$ $\Delta_{1 \rightarrow 2}$

Rekey token

$\{\text{file}_1\}_{k_1}$ $\{\text{file}_2\}_{k_1}$ $\{\text{file}_3\}_{k_1}$ $\{\text{file}_4\}_{k_1}$ $\{\text{file}_5\}_{k_1}$

$\{\text{file}_1\}_{k_2}$ $\{\text{file}_2\}_{k_2}$ $\{\text{file}_3\}_{k_2}$ $\{\text{file}_4\}_{k_2}$
Updatable Encryption

Symmetric Encryption scheme = \((K, E, D)\)

\[ m \xrightarrow{\mathcal{E}} C \xrightarrow{D} m \text{ or } \bot \]

Updatable Encryption scheme = \((K_g, Enc, Dec, RekeyGen, ReEnc)\)

\[ m \xrightarrow{Enc} C_1 \xrightarrow{ReEnc} C_2 \xrightarrow{Dec} m \text{ or } \bot \]
Security Notions

Symmetric Encryption scheme

- Confidentiality: **Ind-Cpa**
  (indistinguishable to chosen-plaintext attack)

- Integrity: **Int-Ctxt**
  (integrity of ciphertext)

Authenticated Encryption:

\[ \text{AE} \Rightarrow \text{Ind-Cpa} \land \text{Int-Ctxt} \]

Updatable Encryption scheme

- Confidentiality: **Up-Ind**

- Integrity: **Up-Int**

Indist. ReEncryption:

**Up-ReEnc**
# Security of Updatable Schemes

<table>
<thead>
<tr>
<th></th>
<th>Confidentiality (Up-Ind)</th>
<th>Integrity (Up-Int)</th>
<th>Indist. ReEncryption (Up-ReEnc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE-hybrid</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>KSS*</td>
<td>✔</td>
<td>✔</td>
<td>X</td>
</tr>
<tr>
<td>[BLMR13]</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ReCrypt*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

* introduced in this work
AE-hybrid is Not Secure

Updatable encryption built with symmetric authenticated encryption (AES-GCM)

\[
\text{Enc}_{k_1}(m): \quad \{x\}^{k_1}_{\text{header}} \cdot \{m\}^{x}_{\text{body}} = C_1
\]

\[
\text{ReEnc}(\Delta, C_1): \quad \{x\}^{k_2} \cdot \{m\}^{x} = C_2
\]

Give the attacker:
\[k_1, \text{ all headers, } C_2\]

Confidentiality (Up-Ind) ×

Integrity (Up-Int) ×

AE-hybrid in production use:

Amazon Web Services

Google Cloud Platform
AE-hybrid Fixed: KSS

**AE-Hybrid**

- $\text{Enc}_{k_1}(m)$
  - $\{x\}_{k_1} \{m\}_x$

**KEM/DEM with Secret Sharing (KSS)**

- $\{x \oplus y, h(m)\}_{k_1}$
  - $y, \{m\}_x$

**Key-share hides $x$ in header**

- Hash gives integrity — binds header/body

- Confidentiality (Up-Ind)
- Integrity (Up-Int)
- Indist. ReEnc (Up-ReEnc)
Strongest Security: ReCrypt

Key Homomorphic Encryption

\[ E_b(E_a(m)) = E_{a \circ b}(m) \quad D_{a \circ b}(E_{a \circ b}(m)) = m \]

\[ d = h(m); \text{ gives integrity} \]

**Enc:**

\[ \{x+y, E_x(d)\}_{k_1} \]

\[ y, E_x(m) \]

**ReEnc:**

\[ \{x'+y'+x+y, E_{x'}(E_x(d))\}_{k_2} \]

\[ y'+y, E_{x'}(E_x(m)) \]

- Confidentiality (Up-Ind)
- Integrity (Up-Int)
- Indist. ReEnc (Up-ReEnc)
Strongest Security Impacts

Performance

<table>
<thead>
<tr>
<th></th>
<th>1 KB</th>
<th>1 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt</td>
<td>10.0 ms</td>
<td>2.6 hrs</td>
</tr>
<tr>
<td>ReEnc</td>
<td>8.8 ms</td>
<td>2.4 hrs</td>
</tr>
<tr>
<td>Decrypt</td>
<td>9.1 ms</td>
<td>2.4 hrs</td>
</tr>
</tbody>
</table>

ReCrypt operations are 1000x slower than KSS

- Good fit for: small, high-value plaintexts
- E.g. credit card numbers, personally-identifying information, financial information
Conclusions

There are significant opportunities for improving the security of internet-scale services.

- **Not-So-Random Numbers [IEEE S&P '14]**
  Environment is fine — entropy rich inputs.
  New designs fix VM reset vulnerabilities; easier to analyze.

- **Pythia PRF Service [Usenix Sec '15]**
  State-of-the-art is broken — new cryptography in service-oriented setting is a great direction.

- **Key Rotation for Auth Encryption [Crypto '17]**
  Customers need updatable encryption — proper balance of security strength and performance is still an open question.