Practical Privacy-Preserving Authentication for SSH

Lawrence Roy
Stanislav Lyakhov
Yeongjin Jang
Mike Rosulek

Oregon State University

To appear at USENIX 2022
ia.cr/2022/740

presentation for IETF 114; 2022-07-25
SSH client  SSH server

should I authenticate with pub key 6c6c6568...?  no

no
SSH client    SSH server

should I authenticate with pub key 6c6c6568...?  no

should I authenticate with pub key 73616664...?  no
SSH client  SSH server

should I authenticate with pub key 6c6c6568...?  
no

should I authenticate with pub key 73616664...?  
no

yes
SSH client        SSH server

should I authenticate with pub key 6c6c6568...?  

no

should I authenticate with pub key 73616664...?  

no

...

yes

signature
SSH client

should I authenticate with pub key 6c6c6568...?  
  no

should I authenticate with pub key 73616664...?  
  no

:

yes

signature

SSH server

problem: server can fingerprint client:

- refuse all advertisements ⇒ learn all keys
Auditing GitHub users’ SSH key quality

A little known feature of GitHub is the ability to look at the public SSH keys that other users have set to be authorised on their account (for example https://github.com/torvalds.keys)

However one of the other side effects of this is that it means that everyone can see your public keys, and if someone cares enough, collect a massive database of everyone’s SSH keys.

Ben Cox https://blog.benjojo.co.uk/post/auditing-github-users-keys
Auditing GitHub users’ SSH key quality

A little known feature of GitHub is the ability to look at the public SSH keys that other users have set to be authorised on their account (for example https://github.com/torvalds.keys)

However one of the other side effects of this is that it means that everyone can see your public keys, and if someone cares enough, collect a massive database of everyone’s SSH keys.

Ben Cox https://blog.benjojo.co.uk/post/auditing-github-users-keys
SSH client

**SSH server**

**problem:** server can fingerprint client:

- I refuse all advertisements
- I learn all keys

**I can configure client to send only “correct” key**

**problem:** client can probe server:

- I order someone else’s pub key, observe response

**SSH supports pre-emptive signature from client**

**problem:** server sees which key was used:

- I and can prove it!

authentication not deniable fundamental to protocol

**problem:** server can act as honeypot:

- I accept any key, even ones never seen before

fundamental to protocol

---

04 Aug 2015

**SSH WHOAMI.FILIPPO.IO**

Here’s a fun PoC I built thanks to [Ben’s dataset](https://words.filippo.io/ssh-whoami-filippo-io/).

I don’t want to ruin the surprise, so just try this command. (It’s harmless.)

```
ssh whoami.filippo.io
```

For the security crowd: don’t worry, I don’t have any OpenSSH 0day and even if I did I wouldn’t burn them on my blog. Also, ssh is designed to log into untrusted servers.

SSH client

ssh whoami.filippo.io

_o/ Hello Mike Rosulek!

Did you know that ssh sends all your public keys to any server it tries to authenticate to?

That's how we know you are @rosulek on GitHub!

Ah, maybe what you didn't know is that GitHub publishes all users' ssh public keys. Myself, I learned it from Ben (benjojo.co.uk).

That's pretty handy at times :) for example your key is at https://github.com/rosulek.keys

-- @FiloSottile (https://twitter.com/FiloSottile)

P.S. The source of this server is at https://github.com/FiloSottile/whoami.filippo.io

Connection to whoami.filippo.io closed.
Did you know that ssh sends all your public keys to any server it tries to authenticate to?

That's how we know you are @Rosulek on GitHub!

Ah, maybe what you didn't know is that GitHub publishes all users' ssh public keys. Myself, I learned it from Ben (benjojo.co.uk).

That's pretty handy at times :) for example your key is at https://github.com/rosulek.keys

-- @FiloSottile (https://twitter.com/FiloSottile)

P.S. The source of this server is at https://github.com/FiloSottile/whoami.filippo.io
SSH client  SSH server

**problem:** server can fingerprint client:

- refuse all advertisements ⇒ learn all keys
- can configure client to send only “correct” key

---

should I authenticate with pub key 6c6c6568...?  

no

should I authenticate with pub key 73616664...?  

no

no

:  

yes  

signature
SSH client | SSH server
---|---
should I authenticate with pub key 6c6c6568...? | no

should I authenticate with pub key 73616664...? | no

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
</tr>
<tr>
<td>signature</td>
</tr>
</tbody>
</table>

**problem:** server can fingerprint client:
- refuse all advertisements \(\implies\) learn all keys
- can configure client to send only “correct” key

**problem:** client can probe server:
- offer someone else’s pub key, observe response
- SSH supports *pre-emptive* signature from client
SSH client: should I authenticate with pub key 6c6c6568...?  
no

SSH client: should I authenticate with pub key 73616664...?  
o

SSH server: problem: server can fingerprint client:
- refuse all advertisements ⇒ learn all keys
- can configure client to send only “correct” key

SSH server: problem: client can probe server:
- offer someone else’s pub key, observe response
- SSH supports pre-emptive signature from client

SSH server: problem: server sees which key was used:
- and can prove it! ⇒ authentication not deniable
- fundamental to protocol
**SSH client** | **SSH server**
---|---
should I authenticate with pub key 6c6c6568...? | no

should I authenticate with pub key 73616664...? | no

: |

yes | signature

**problem:** server can fingerprint client:
- refuse all advertisements ⇒ learn all keys
- can configure client to send only “correct” key

**problem:** client can probe server:
- offer someone else’s pub key, observe response
- SSH supports *pre-emptive* signature from client

**problem:** server sees which key was used:
- and can **prove it!** ⇒ authentication not deniable
- fundamental to protocol

**problem:** server can act as honeypot:
- accept *any* key, even ones never seen before
- fundamental to protocol
our new authentication method: big picture

can include any mixture of existing RSA, ECDSA, EdDSA keys, in a single authentication attempt
our new authentication method: big picture

client

$sk_1, sk_4, sk_9$

server

$pk_1, pk_2, \ldots, pk_6$

our protocol

client has 3 keys, including

at least one of \{$sk_1, \ldots, sk_6$\}

can include any mixture of existing RSA, ECDSA, EdDSA keys, in a single authentication attempt
our new authentication method: big picture

server has 6 keys, including \( pk_1 \) and \( pk_4 \)

client has 3 keys, including \( sk_1 \) and at least one of \( \{sk_1, \ldots, sk_6\} \)

- can include any mixture of existing RSA, ECDSA, EdDSA keys, in a single authentication attempt
our new authentication method: big picture

- Our protocol
  - client: \( sk_1, sk_4, sk_9 \)
  - server: \( pk_1, pk_2, \ldots, pk_6 \)

- Server has 6 keys, including \( pk_1 \) and \( pk_4 \)
- Client has 3 keys, including at least one of \( \{ sk_1, \ldots, sk_6 \} \)

- Can include any mixture of existing RSA, ECDSA, EdDSA keys, in a single authentication attempt
- Does not depend on site-specific configuration; both parties can just safely use all their keys
our new authentication method: big picture

- **client**
  - $sk_1, sk_4, sk_9$

- **server**
  - $pk_1, pk_2, \ldots, pk_6$

Server has 6 keys, including $pk_1$ and $pk_4$

Client has 3 keys, including at least one of $\{sk_1, \ldots, sk_6\}$

- can include any mixture of existing RSA, ECDSA, EdDSA keys, in a single authentication attempt
- does not depend on site-specific configuration; both parties can just safely use all their keys
- client won’t connect unless server knows and explicitly includes one of client’s keys
technical overview

client (with $\{sk_i\}_i$):  

server (with $\{pk_j\}_j$):

1. anonymous multi-KEM
   $c = \text{Dec}^1 sk_i \cdot c \circ \text{PSI} \text{fmj}$
   Each party has a set of items; client learns the intersection; server learns whether it is empty.

2. private set intersection
   $fbmi = \text{?}$
   Single MKEM construction supporting RSA, ECDSA, & EdDSA
   Add "proof of nonempty intersection" to [RosulekTrieu21] PSI + full UC security analysis.
technical overview

client (with \( \{sk_i\}_i\)):

server (with \( \{pk_j\}_j\)):

\[
c, \{m_j\}_j \leftarrow \text{Enc}\left(\{pk_j\}_j\right)
\]

1. anonymous multi-KEM

address ciphertext to \( \{pk_j\}_j\);
\( sk_j \) decrypts \( c \) to \( m_j \);
\( c \) hides \( pk_j \) recipients

\[\text{PSI} = \text{PSI}^+ \] full UC security analysis
technical overview

client (with \( \{sk_i\}_i\)):

\[ c \]

server (with \( \{pk_j\}_j\)):

\[ c, \{m_j\}_j \leftarrow \text{Enc}\left(\{pk_j\}_j\right) \]

1. anonymous multi-KEM

- address ciphertext to \( \{pk_j\}_j \);
- \( sk_j \) decrypts \( c \) to \( m_j \);
- \( c \) hides \( pk_j \) recipients

\[ \text{PSI} \]

\[ \text{add "proof of nonempty intersection" to [RosulekTrieu21]} \]

\[ \text{full UC security analysis} \]
client (with \( \{sk_i\}_i\)):

\[
\{ \tilde{m}_i := \text{Dec}(sk_i, c) \}_i
\]

server (with \( \{pk_j\}_j\)):

\[
c, \{m_j\}_j \leftarrow \text{Enc}(\{pk_j\}_j)
\]

1. anonymous multi-KEM

address ciphertext to \( \{pk_j\}_j\);

\( sk_j \) decrypts \( c \) to \( m_j \);

\( c \) hides \( pk_j \) recipients

\( \text{PSI} \) for each party has set of items;
client learns intersection;
server learns whether empty

\( \text{PSI}^+ \) full UC security analysis

[RosulekTrieu21]
1. anonymous multi-KEM

- address ciphertext to $\{pk_j\}_j$;
- $sk_j$ decrypts $c$ to $m_j$;
- $c$ hides $pk_j$ recipients

2. private set intersection

- each party has set of items;
technical overview

**client (with \{sk_i\}_i):**

\[
\{\widehat{m}_i := \text{Dec}(sk_i, c)\}_i
\]

**server (with \{pk_j\}_j):**

\[
c, \{m_j\}_j \leftarrow \text{Enc}\left(\{pk_j\}_j\right)
\]

1. **anonymous multi-KEM**

   - address ciphertext to \{pk_j\}_j;
   - \(sk_j\) decrypts \(c\) to \(m_j\);
   - \(c\) hides \(pk_j\) recipients

2. **private set intersection**

   - each party has set of items;
   - client learns intersection;
technical overview

client (with \( \{sk_i\}_i\)):

\[
\{\hat{m}_i := \text{Dec}(sk_i, c)\}_i
\]

server (with \( \{pk_j\}_j\)):

\[
c, \{m_j\}_j \leftarrow \text{Enc}\left(\{pk_j\}_j\right)
\]

1. anonymous multi-KEM

address ciphertext to \( \{pk_j\}_j \);

\( sk_j \) decrypts \( c \) to \( m_j \);

\( c \) hides \( pk_j \) recipients

2. private set intersection

each party has set of items;

client learns intersection;

server learns whether empty

\[
\{\hat{m}_i\}_i \cap \{m_j\}_j
\]

\[
\cap = \emptyset?
\]
technical overview & contributions

client (with $\{sk_i\}_i$):

$$\{\tilde{m}_i := \text{Dec}(sk_i, c)\}_i$$

server (with $\{pk_j\}_j$):

$$c, \{m_j\}_j \leftarrow \text{Enc}\left(\{pk_j\}_j\right)$$

1. anonymous multi-KEM

single MKEM construction supporting RSA, ECDSA, & EdDSA

2. private set intersection

each party has set of items; client learns intersection; server learns whether empty
### Technical Overview & Contributions

**Client (with \( \{sk_i\}_i\))**:

- \( \{\hat{m}_i := \text{Dec}(sk_i, c)\}_i \)

**Server (with \( \{pk_j\}_j\))**:

- \( c, \{m_j\}_j \leftarrow \text{Enc}(\{pk_j\}_j) \)

1. **Anonymous Multi-KEM**
   - Single MKEM construction supporting RSA, ECDSA, & EdDSA
   - Add "proof of nonempty intersection" to [RosulekTrieu21] PSI

2. **Private Set Intersection**
   - \( \{\hat{m}_i\}_i \cap \{m_j\}_j \)
   - \( \cap = \emptyset ? \)
technical overview & contributions

client (with \( \{sk_i\}_i\)):  
\[
\{\widehat{m}_i := \text{Dec}(sk_i, c)\}_i
\]

server (with \( \{pk_j\}_j\)):  
\[
c, \{m_j\}_j \leftarrow \text{Enc}\left(\{pk_j\}_j\right)
\]

1. anonymous multi-KEM  
   single MKEM construction supporting RSA, ECDSA, & EdDSA

2. private set intersection  
   add “proof of nonempty intersection” to [RosulekTrieu21] PSI

+ full UC security analysis
**concrete performance:**

<table>
<thead>
<tr>
<th># of keys</th>
<th>RSA keys only</th>
<th>EC keys only</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td>server</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>60 ms</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>320 ms</td>
</tr>
<tr>
<td>20</td>
<td>1000</td>
<td>1200 ms</td>
</tr>
</tbody>
</table>

2 commodity desktop computers on LAN
## concrete performance:

<table>
<thead>
<tr>
<th># of keys</th>
<th>RSA keys only</th>
<th>EC keys only</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td>server</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>60 ms</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>320 ms</td>
</tr>
<tr>
<td>20</td>
<td>1000</td>
<td>1200 ms</td>
</tr>
<tr>
<td>20</td>
<td>1000</td>
<td>214 ms</td>
</tr>
</tbody>
</table>

2 commodity desktop computers on LAN
**Concrete performance:**

<table>
<thead>
<tr>
<th># of keys</th>
<th>RSA keys only</th>
<th>EC keys only</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td>server</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>60 ms</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>9 ms</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>320 ms</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>28 ms</td>
</tr>
</tbody>
</table>

2 commodity desktop computers on LAN
**concrete performance:**

<table>
<thead>
<tr>
<th># of keys</th>
<th>RSA keys only</th>
<th>EC keys only</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td>server</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>60 ms</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>320 ms</td>
</tr>
<tr>
<td>20</td>
<td>1000</td>
<td>1200 ms</td>
</tr>
</tbody>
</table>

2 commodity desktop computers on LAN
our protocol

client

set of secret keys

# of server keys;

identity of authorized keys

server

set of “authorized” public keys

# of client keys;

were any of them authorized?