SSH client  SSH server

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

no

no

"yes"

"signature"

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 over someone else's pub key, observe response
I pre-emptive signatures possible (in principle)

problem:
server sees which key was used:
I and can prove it!
I fundamental to protocol

problem:
server can act as honeypot:
I accept any key, even ones never seen before
I fundamental to protocol
SSH client    SSH server

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

no

should I authenticate
with pub key 73616664...?

no

署名问题:
服务器可以追踪客户端:
我拒绝所有广告
学习所有密钥
我可以配置客户端只发送“正确”的密钥

署名问题:
客户端可以探测服务器:
我允许其他人使用公钥，观察响应
预支性署名（在原则上）可能

署名问题:
服务器可以看到使用的密钥:
我用它来证明它！

认证不可否认

基本协议

署名问题:
服务器可以充当诱饵:
我接受任何密钥，即使从未见过的密钥

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

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

| : | yes |
should I authenticate with pub key 6c6c6568...?
no

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

::

yes

signature
SSH client  SSH server

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

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

:::

problem: server can fingerprint client:
  ➤ refuse all advertisements ⇒ learn all keys
SSH client

SSH server

SSH WHOAMI.FILIPPO.IO

Here's a fun PoC I built thanks to Ben's dataset.

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

```bash
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.

Filippo Valsorda https://words.filippo.io/ssh-whoami-filippo-io/
SSH client

```bash
[kochanski:]$ 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.
```
SSH client

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

no

should I authenticate with pub key 73616664...?

no

"yes signature problem:
server can ngerprint client:
I refuse all advertisements
I learn all keys
I can congure client to send only "correct" key

I can preemptive signatures possible (in principle)

problem:
server sees which key was used:
I and can prove it!

authentication not deniable
I fundamental to protocol

problem:
server can act as honeypot:
I accept any key, even ones never seen before
I fundamental to protocol

SSH server

[kochanski:~]$ 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.
SSH client          SSH server

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

no

should I authenticate with pub key 73616664...?  

no

no

:**problem:** server can fingerprint client:

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

problem: server can act as honeypot:

- accept any key, even ones never seen before
- fundamental to protocol
SSH client | SSH server
--- | ---
should I authenticate with Bob’s pub key? | yes/no

**Problem:** server can fingerprint client:
- refuse all advertisements $\Rightarrow$ learn all keys
- can configure client to send only “correct” key

**Problem:** client can probe server:
- offer someone else’s pub key, observe response
- *pre-emptive* signatures possible (in principle)
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
- *pre-emptive* signatures possible (in principle)

**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...?

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

**problem:** client can probe server:
- offer someone else’s pub key, observe response
- *pre-emptive* signatures possible (in principle)

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

**problem:** server can act as honeypot:
- accept *any* key, even ones never seen before
- fundamental to protocol
goals of this work

1 server & client should learn minimal information
goals of this work

1. server & client should learn minimal information
2. authenticate with respect to existing SSH keys
goals of this work

1. server & client should learn minimal information
2. authenticate with respect to existing SSH keys
3. minimize reliance on per-site configuration

https://github.blog/2021-09-01-improving-git-protocol-security-github/
goals of this work

1. server & client should learn minimal information
2. authenticate with respect to existing SSH keys
3. minimize reliance on per-site configuration
our new authentication method: big picture

client

\[ s_{k1}, s_{k4}, s_{k9} \]

server

\[ p_{k1}, p_{k2}, \ldots, p_{k6} \]

our protocol

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: \( pk_1, pk_2, \ldots, pk_6 \)

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

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

- **client**
  - sk₁, sk₄, sk₉
- **server**
  - pk₁, pk₂, ..., pk₆

- **our protocol**

  - server has 6 keys, including pk₁ and pk₄
  - client has 3 keys, including at least one of {sk₁, ..., sk₆}

- 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 at least one of \( \{sk_1, \ldots, sk_6\} \)

- any mixture of existing RSA, ECDSA, EdDSA keys, in a single authentication attempt
- does not depend on site-specific configuration; safe to use all keys in every authentication attempts
our new authentication method: big picture

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

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

- any mixture of existing RSA, ECDSA, EdDSA keys, in a single authentication attempt
- does not depend on site-specific configuration; safe to use all keys in every authentication attempts
- client won’t connect unless server knows and explicitly includes one of client’s keys
technical overview

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

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

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

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

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

1. anonymous multi-KEM

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

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

\( c \) hides \( pk_j \) recipients
technical overview

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

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

1. anonymous multi-KEM

address ciphertext to \{pk_j\}_j;
sk_j decrypts c to \{m_j\}_j;
c hides pk_j recipients
technical overview

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

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

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

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

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

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

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

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

1. anonymous multi-KEM

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

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

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

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

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

1. anonymous multi-KEM

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

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

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

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

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

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

1. anonymous multi-KEM
   - single MKEM construction supporting RSA, ECDSA, & EdDSA
   - add “proof of nonempty intersection” to [RosulekTrieu21] PSI

2. private set intersection
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

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

+ full UC security analysis
**concrete performance (in OpenSSH):**

<table>
<thead>
<tr>
<th># of keys</th>
<th>RSA keys only (worst case for us)</th>
<th>{EC,Ed}DSA keys only (best case for us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td>server</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>comm</td>
<td>time</td>
</tr>
<tr>
<td>comm</td>
<td></td>
<td>comm</td>
</tr>
</tbody>
</table>

- RSA keys only
- {EC,Ed}DSA keys only

github.com/osu-crypto/PSIPK-ssh

2 commodity desktop computers on LAN
concrete performance (in OpenSSH):

<table>
<thead>
<tr>
<th># of keys</th>
<th>RSA keys only (worst case for us)</th>
<th>{EC,Ed}DSA keys only (best case for us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td>server</td>
<td>time</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>60 ms</td>
</tr>
</tbody>
</table>

github.com/osu-crypto/PSIPK-ssh 2 commodity desktop computers on LAN
## concrete performance (in OpenSSH):

<table>
<thead>
<tr>
<th># of keys</th>
<th>RSA keys only</th>
<th>{EC,Ed}DSA keys only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time comm</td>
<td>time comm</td>
</tr>
<tr>
<td>client</td>
<td>server</td>
<td>(worst case for us)</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>60 ms 12 kB</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>320 ms 53 kB</td>
</tr>
</tbody>
</table>

2 commodity desktop computers on LAN

github.com/osu-crypto/PSIPK-ssh
**concrete performance (in OpenSSH):**

<table>
<thead>
<tr>
<th># of keys</th>
<th>RSA keys only</th>
<th>{EC,Ed}DSA keys only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(worst case for us)</td>
<td>(best case for us)</td>
</tr>
<tr>
<td>client</td>
<td>server</td>
<td>time</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>

github.com/osu-crypto/PSIPK-ssh

2 commodity desktop computers on LAN
set of secret keys

set of "authorized" public keys

# of server keys; identity of authorized keys

# of client keys; were any of them authorized?

✓ efficient, practical
✓ mixture of existing RSA & EC keys
✓ safe without special per-site configuration
client

set of secret keys

# of server keys;

identity of authorized keys

our protocol

server

set of “authorized” public keys

# of client keys;

were any of them authorized?

✓ efficient, practical
✓ mixture of existing RSA & EC keys
✓ safe without special per-site configuration

thanks!

github.com/osu-crypto/PSIPK-ssh

ia.cr/2022/740
(backup slides)
github over SSH:

client → github.com

authenticate server

username = git

negotiate choice of pk

authenticate

commit to repositoryname

server must decide set of authorized keys before running our protocol!

server does not know repository name yet!

use repository name as username
github over SSH:

Client 

authenticate server

username = git

negotiate choice of pk

authenticate

commit to repositoryname

Server

github.com

our protocol

I

set of authorized keys

before running our protocol!

I

use repository name as username

ourprotocol

??

??
github over SSH:

- client to github.com
- authenticate server
- username = git
- negotiate choice of pk
- authenticate
- commit to repositoryname

- server must decide set of authorized keys before running our protocol!
github over SSH:

- client
- authenticate server
- username = git
- negotiate choice of pk
- authenticate
- commit to repository

- server must decide set of authorized keys before running our protocol!
- server does not know repository name yet!
github over SSH:

client new.github.com

authenticate server

username = repositoryname

negotiate choice of pk

our protocol ✓

authenticate

commit

- server must decide set of authorized keys before running our protocol!
- server does not know repository name yet!
- use repository name as username
1. anonymous multi-KEM

address ciphertext to \(\{pk_j\}_j\);
\(sk_j\) decrypts \(c\) to \(m_j\);
\(c\) hides \(pk_j\) recipients
the case of EdDSA/ECDSA

Alice: $pk_A = g^a$
Bob: $pk_B = g^b$
Charlie: $pk_C = g^c$

1. anonymous multi-KEM
   address ciphertext to $\{pk_j\}_j$; $sk_j$ decrypts $c$ to $m_j$; $c$ hides $pk_j$ recipients
the case of EdDSA/ECDSA

Alice: $pk_A = g^a$
Bob: $pk_B = g^b$
Charlie: $pk_C = g^c$

ciphertext = $g^r$

1. anonymous multi-KEM

address ciphertext to $\{pk_j\}_j$;
skj decrypts c to $m_j$;
c hides $pk_j$ recipients
the case of EdDSA/ECDSA

Alice: $pk_A = g^a$
Bob: $pk_B = g^b$
Charlie: $pk_C = g^c$

ciphertext = $g^r$

Alice will decrypt to $(pk_A)^r$
Bob will decrypt to $(pk_B)^r$
Charlie will decrypt to $(pk_C)^r$

1. anonymous multi-KEM

address ciphertext to $\{pk_j\}_j$;
$sk_j$ decrypts $c$ to $m_j$;
c hides $pk_j$ recipients
Alice: $pk_A = g^a$
Bob: $pk_B = g^b$
Charlie: $pk_C = g^c$

ciphertext = $g^r$

Alice will decrypt to $(pk_A)^r$
Bob will decrypt to $(pk_B)^r$
Charlie will decrypt to $(pk_C)^r$

ciphertext hides set of recipients; even # of them!
The case of RSA

Alice: $pk_A = (N_A, e_A)$
Bob: $pk_B = (N_B, e_B)$
Charlie: $pk_C = (N_C, e_C)$

1. anonymous multi-KEM

address ciphertext to $\{pk_j\}_j$;
$sk_j$ decrypts $c$ to $m_j$;
c hides $pk_j$ recipients
the case of RSA

Alice: \( pk_A = (N_A, e_A) \)
Bob: \( pk_B = (N_B, e_B) \)
Charlie: \( pk_C = (N_C, e_C) \)

encrypt \((r_A)^{e_A} \mod N_A\)
encrypt \((r_B)^{e_B} \mod N_B\)
encrypt \((r_C)^{e_C} \mod N_C\)

1. anonymous multi-KEM

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

Alice: $pk_A = (N_A, e_A)$
Bob: $pk_B = (N_B, e_B)$
Charlie: $pk_C = (N_C, e_C)$

encrypt $(r_A)^{e_A}$ mod $N_A$
encrypt $(r_B)^{e_B}$ mod $N_B$
encrypt $(r_C)^{e_C}$ mod $N_C$

interpolate poly $P$:

$P(N_A) = (r_A)^{e_A}$
$P(N_B) = (r_B)^{e_B}$
$P(N_C) = (r_C)^{e_C}$

1. anonymous multi-KEM

address ciphertext to $\{pk_j\}_j$;
$sk_j$ decrypts $c$ to $m_j$;
c hides $pk_j$ recipients
the case of RSA

Alice: $pk_A = (N_A, e_A)$
Bob: $pk_B = (N_B, e_B)$
Charlie: $pk_C = (N_C, e_C)$

encrypt $(r_A)^{e_A} \mod N_A$
encrypt $(r_B)^{e_B} \mod N_B$
encrypt $(r_C)^{e_C} \mod N_C$

interpolate poly $P$:
$P(N_A) = (r_A)^{e_A}$
$P(N_B) = (r_B)^{e_B}$
$P(N_C) = (r_C)^{e_C}$

ciphertext = $P$

1. anonymous multi-KEM

address ciphertext to $\{pk_j\}_j$;
sk$_j$ decrypts $c$ to $m_j$;
c hides $pk_j$ recipients
PSI with proof of nonempty intersection

2. **private set intersection**

   each party has set of items;
   client learns intersection;
   server learns whether empty
oblivious PRF (OPRF) paradigm for PSI

[FreedmanIshaiPinkasReingold05]

Alice:
\[ X = \{ x_1, x_2, \ldots \} \]

Bob:
\[ Y = \{ y_1, y_2, \ldots \} \]
oblivious PRF (OPRF) paradigm for PSI

[FreedmanIshaiPinkasReingold05]

Alice: $X = \{x_1, x_2, \ldots\}$

Bob: $Y = \{y_1, y_2, \ldots\}$

$\overline{\text{OPRF}}$
oblivious PRF (OPRF) paradigm for PSI

[FreedmanIshaiPinkasReingold05]

Alice:

\[ X = \{x_1, x_2, \ldots \} \]

Bob:

\[ Y = \{y_1, y_2, \ldots \} \]

\[ F(x_1), F(x_2), \ldots \]
### oblivious PRF (OPRF) paradigm for PSI

[FreedmanIshaiPinkasReingold05]

- **Alice:**
  \[ X = \{ x_1, x_2, \ldots \} \]

- **Bob:**
  \[ Y = \{ y_1, y_2, \ldots \} \]

\[ F(x_1), F(x_2), \ldots \quad \text{OPRF} \quad \text{random } F(\cdot) \]

\[ F(y_1), F(y_2), \ldots \]
oblivious PRF (OPRF) paradigm for PSI

Alice:
\[ X = \{x_1, x_2, \ldots\} \]

Bob:
\[ Y = \{y_1, y_2, \ldots\} \]

\[ F(x_1), F(x_2), \ldots \]

random \( F(\cdot) \)

\[ F(y_1), F(y_2), \ldots \]
**oblivious PRF (OPRF) paradigm for PSI**

[FreedmanIshaiPinkasReingold05]

\[ \text{Alice:} \quad X = \{x_1, x_2, \ldots \} \]

\[ \text{Bob:} \quad Y = \{y_1, y_2, \ldots \} \]

\[ x_1, x_2, \ldots \]

\[ F(x_1), F(x_2), \ldots \]

\[ F(y_1), F(y_2), \ldots \]

\[ \text{OPRF} \]

\[ \text{random } F(\cdot) \]

\[ \text{Enc}(F(x_1), r), \text{Enc}(F(x_2), r), \ldots \]

\[ H(r) \]
overtive PRF (OPRF) paradigm for PSI

[FreedmanIshaiPinkasReingold05]

Alice:
\[ X = \{x_1, x_2, \ldots \} \]
\[ x_1, x_2, \ldots \]
\[ F(x_1), F(x_2), \ldots \]

Bob:
\[ Y = \{y_1, y_2, \ldots \} \]
\[ \text{random } F(\cdot) \]
\[ F(y_1), F(y_2), \ldots \]
\[ \text{Enc}(F(x_1), r), \text{Enc}(F(x_2), r), \ldots \]
\[ H(r) \]
\[ r \]