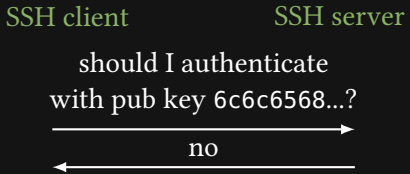
Practical Privacy-Preserving Authentication for SSH

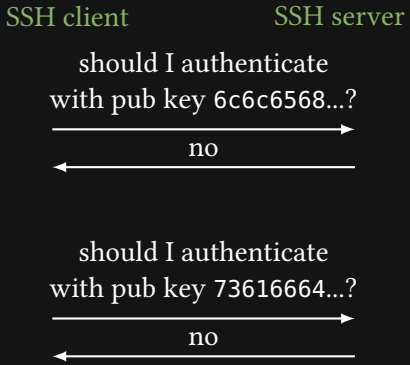
Lawrence Roy
Stanislav Lyakhov
Yeongjin Jang
Mike Rosulek

Oregon State University

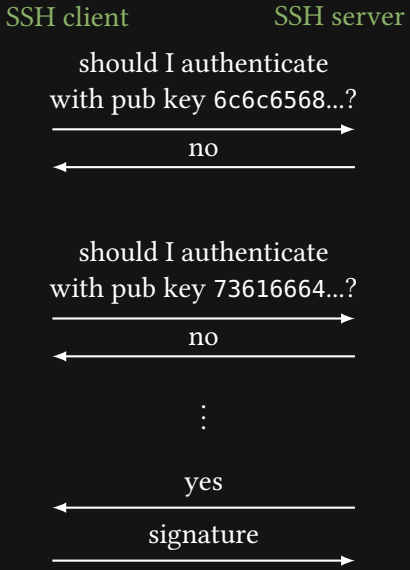
USENIX Security 2022
ia.cr/2022/740

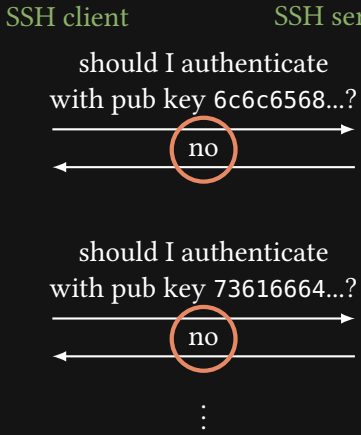
2022-08-12


SSH client
SSH server
should I authenticate
with pub key 6c6c6568...?
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

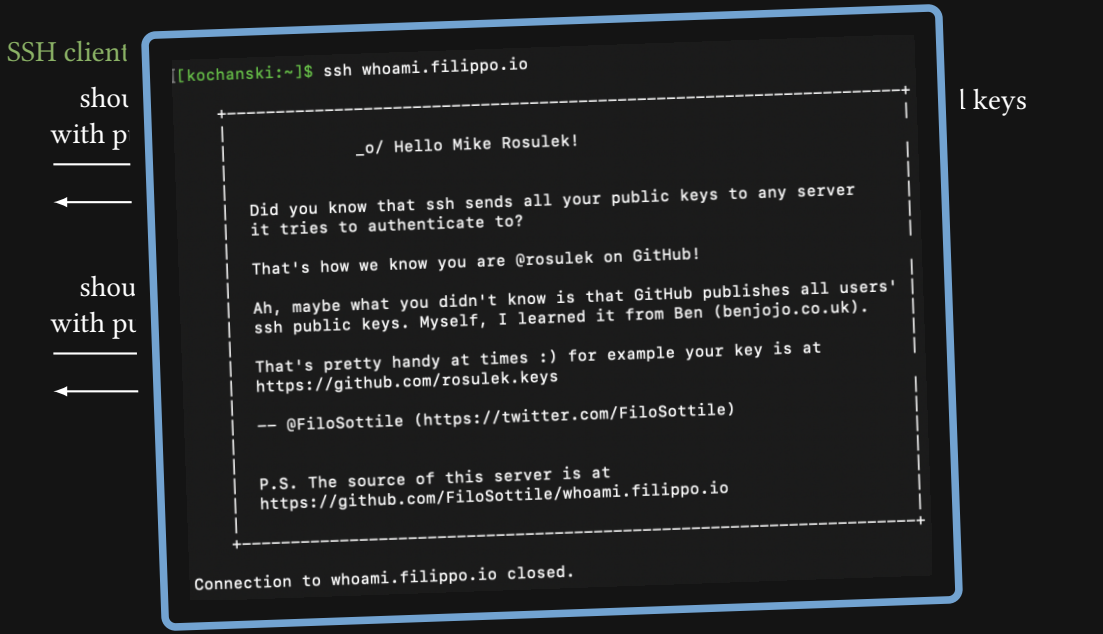

SSH client
SSH server
should I authenticate with pub key 6c6c6568...?
no
should I authenticate with pub key 73616664...?
no
yes
signature

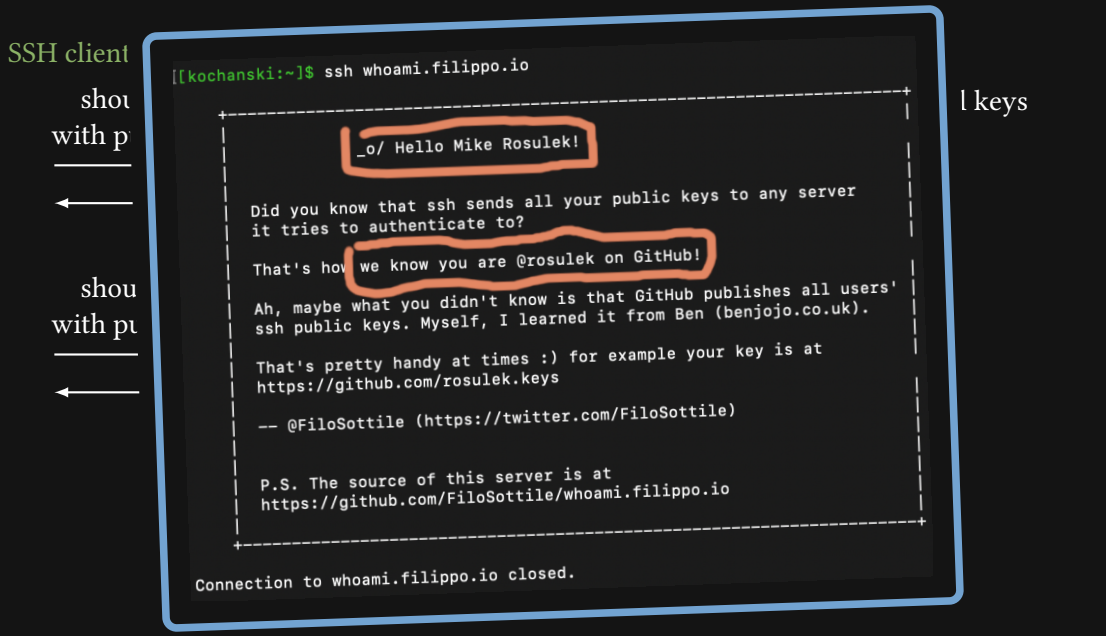


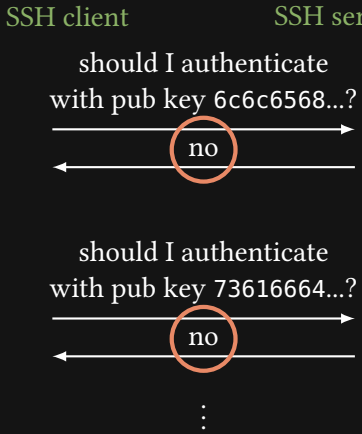
problem: server can fingerprint client:

- refuse all advertisements ⟹ learn all keys


SSH client
SSH server
problem:
shou
with p
shou
with pu
l keys
04 Aug 2015
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.)
ssh whoami.filippo.io
For the security crowd: don't worry, I don't have any OpenSSH oday 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/

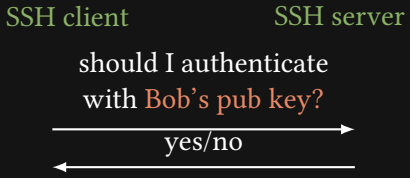

SSH client
shou
with p
shou
with pu
l keys
[[kochanski:~]$ ssh whoami.filippo.io
_o/ Hello Mike Rosulek!
Did you know that ssh sends all your public keys to any server
it tries to authenticate to?
That's how we know you are @rosulek on GitHub!
Ah, maybe what you didn't know is that GitHub publishes all 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
shou
with p
shou
with pu
l keys
[[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 hov 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.



problem: server can fingerprint client:

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



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 can fingerprint client:

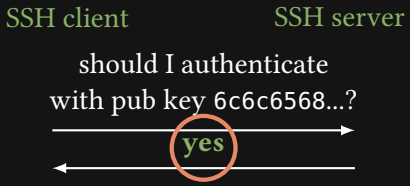
- refuse all advertisements ⇒ learn all keys
- can configure client to send only "correct" key

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



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

problem: server can act as honeypot:

- accept *any* key, even ones never seen before
- fundamental to protocol

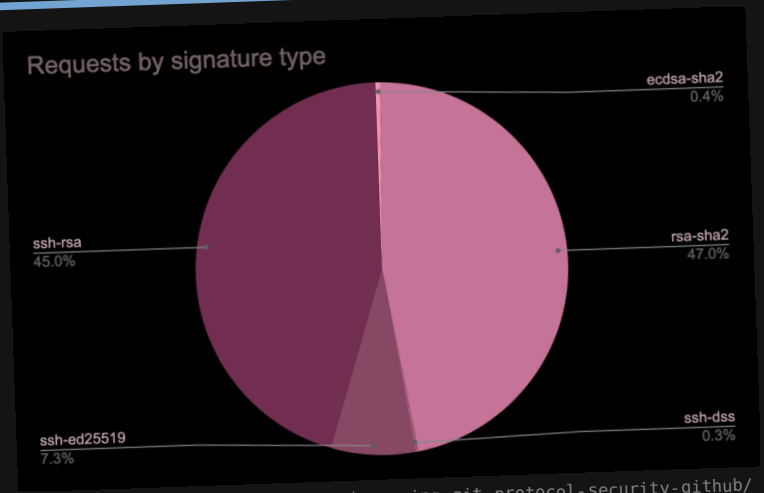
goals of this work

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

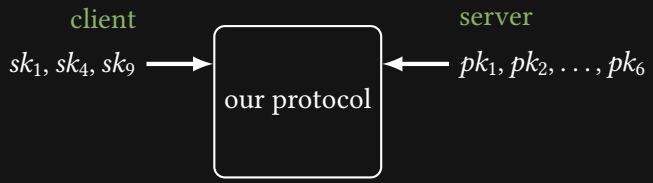


https://github.blog/2021-09-01-improving-git-protocol-security-github/

goals of this work

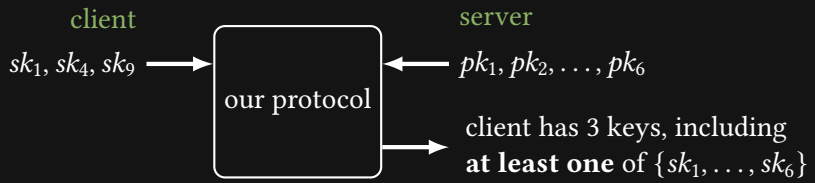
1. server & client should learn minimal information
2. authenticate with respect to existing SSH keys
3. minimize reliance on per-site configuration

our new authentication method: big picture



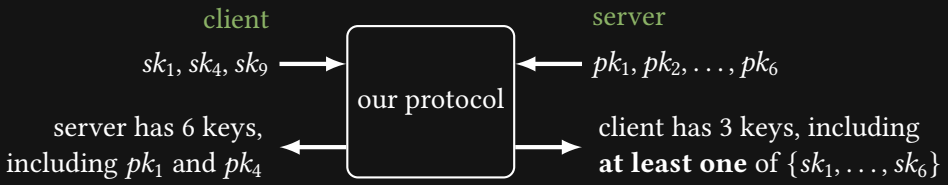
- any **mixture** of existing RSA, ECDSA, EdDSA keys, in a single authentication attempt

our new authentication method: big picture



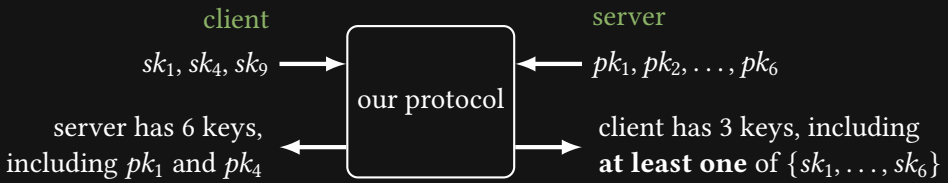
- any **mixture** of existing RSA, ECDSA, EdDSA keys,
  in a single authentication attempt

our new authentication method: big picture



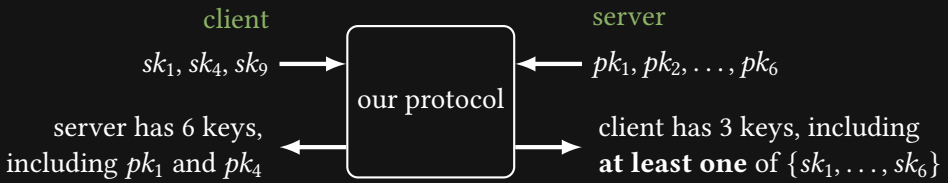
- any **mixture** of existing RSA, ECDSA, EdDSA keys,
  in a single authentication attempt

our new authentication method: big picture



- 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



- 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 $\{pk_j\}_j$):

$c, \{m_j\}_j \leftarrow \mathsf{Enc}\Big(\{pk_j\}_j\Big)$

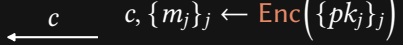
1. anonymous multi-KEM

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

technical overview

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

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



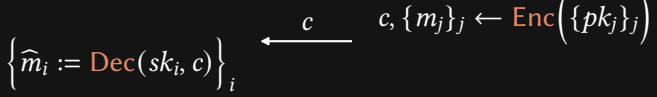
1. anonymous multi-KEM

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

technical overview

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

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



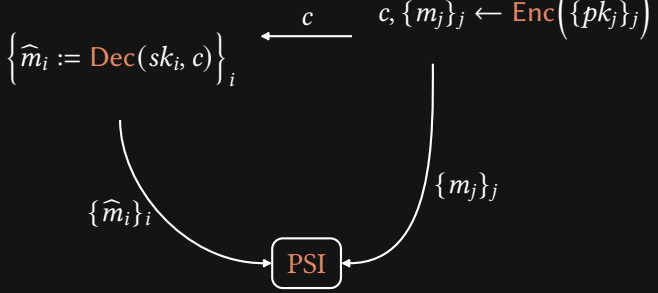
1. anonymous multi-KEM

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

technical overview

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

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

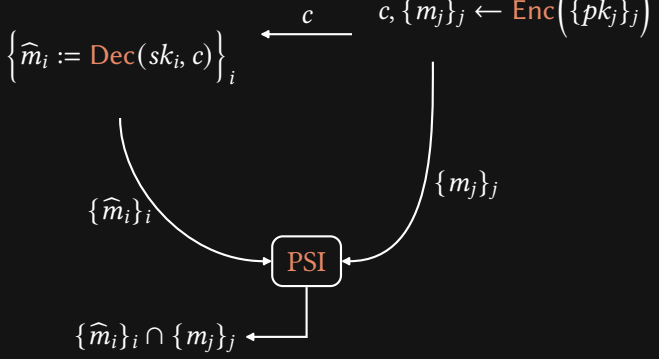
2. private set intersection

each party has set of items;

technical overview

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

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



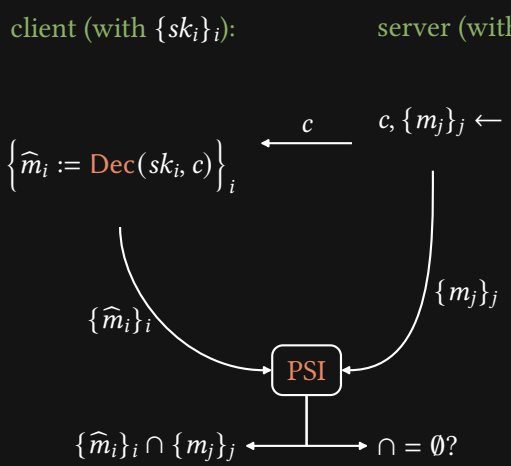
1. anonymous multi-KEM

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

2. private set intersection

each party has set of items;
client learns intersection;

technical overview



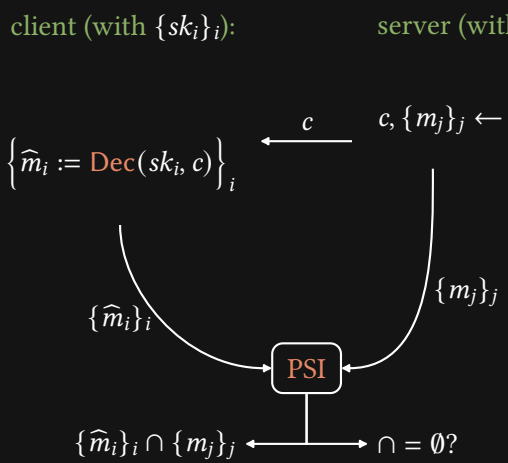
1. anonymous multi-KEM

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 learns intersection;
server learns whether empty

technical overview & contributions



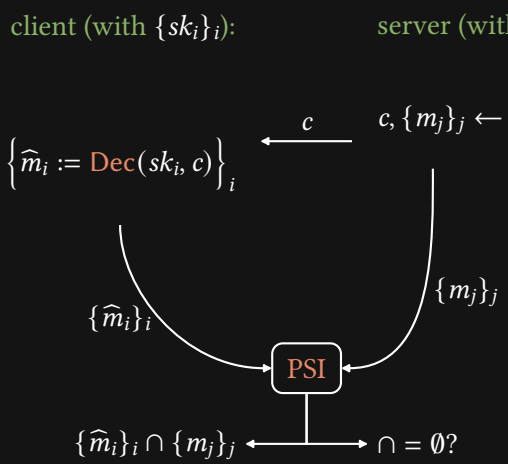
1. anonymous multi-KEM

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



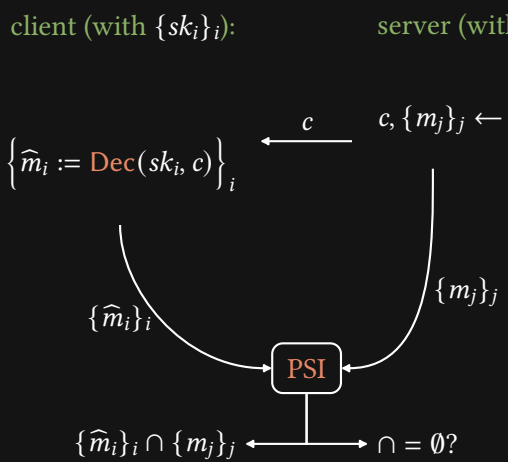
1. anonymous multi-KEM

single MKEM construction supporting RSA, ECDSA, & EdDSA

2. private set intersection

add "proof of nonempty intersection" to [RosulekTrieu21] PSI

technical overview & contributions



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):

# of keys		RSA keys only (worst case for us)		{EC,Ed}DSA keys only (best case for us)	
client	server	time	comm	time	comm

`github.com/osu-crypto/PSIPK-ssh`

2 commodity desktop computers on LAN

concrete performance (in OpenSSH):

# of keys		RSA keys only (worst case for us)		{EC,Ed}DSA keys only (best case for us)	
client	server	time	comm	time	comm
5	10	60 ms	12 kB	9 ms	8 kB

`github.com/osu-crypto/PSIPK-ssh`

2 commodity desktop computers on LAN

concrete performance (in OpenSSH):

# of keys		RSA keys only (worst case for us)		{EC,Ed}DSA keys only (best case for us)	
client	server	time	comm	time	comm
5	10	60 ms	12 kB	9 ms	8 kB
20	100	320 ms	53 kB	28 ms	12 kB

`github.com/osu-crypto/PSIPK-ssh`

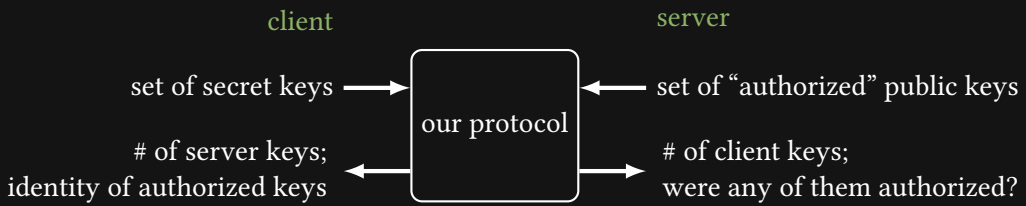
2 commodity desktop computers on LAN

concrete performance (in OpenSSH):

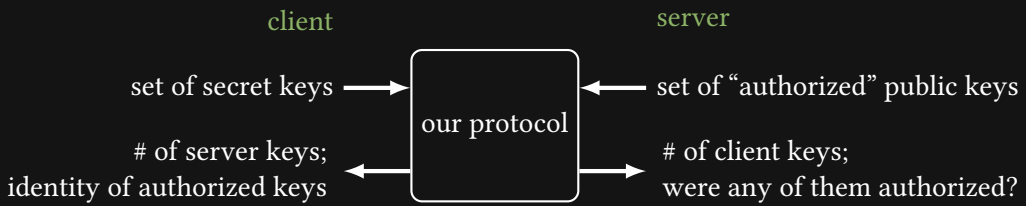
# of keys		RSA keys only (worst case for us)		{EC,Ed}DSA keys only (best case for us)	
client	server	time	comm	time	comm
5	10	60 ms	12 kB	9 ms	8 kB
20	100	320 ms	53 kB	28 ms	12 kB
20	1000	1200 ms	460 kB	214 ms	41 kB

`github.com/osu-crypto/PSIPK-ssh`

2 commodity desktop computers on LAN



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



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

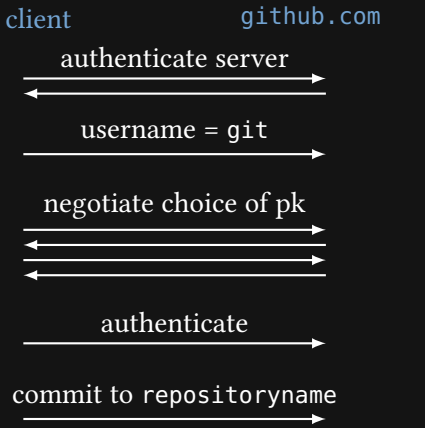
thanks!

`github.com/osu-crypto/PSIPK-ssh`

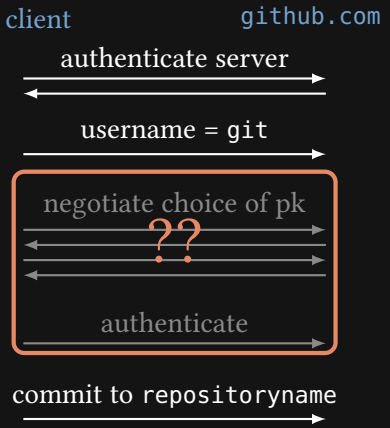
`ia.cr/2022/740`

(backup slides)

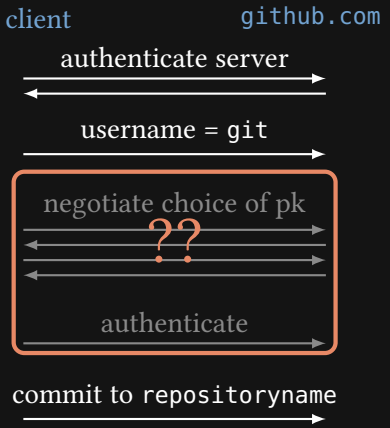
github over SSH:



github over SSH:

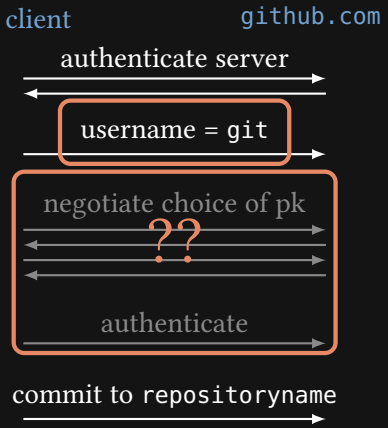


github over SSH:



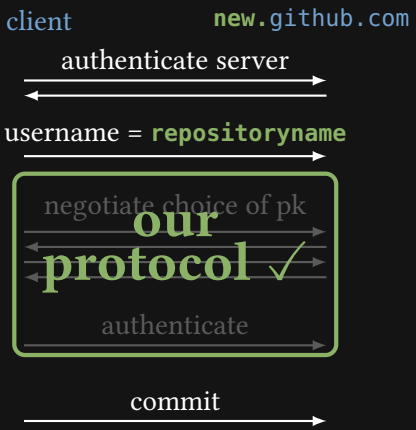
- server must decide **set of authorized keys** before running our protocol!

github over SSH:



- server must decide **set of authorized keys** before running our protocol!
- server does not know repository name yet!

github over SSH:



- server must decide **set of authorized keys** before running our protocol!
- server does not know repository name yet!
- use repository name as `username`