## **TLS** And some stuff about making it Post-Quantum

Thom Wiggers, 2022-06-01, Radboud University - Applied Cryptography



### Of US Firefox page loads use TLS

Firefox Telemetry, 2022-05-30



## **Agenda** What are we talking about today

- TLS
  - History
  - TLS 1.3
- PKI
- Making stuff PQ
  - PQTLS
  - KEMTLS

## "TLS allows client/server applications to communicate over the Internet in a way that is designed to prevent eavesdropping, tampering, and message forgery."

**RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3** 

○ A https://thomwiggers.nl  $\leftarrow$ С  $\rightarrow$ 

Thom Wiggers Home Posts Talks Publications Teaching Docs Contact



Thom Wiggers

PhD candidate at Radboud University

Radboud University



Thom Wiggers is a PhD Candidate at the Institute of Computing and Information Sciences, Radboud University in The Netherlands. He is working on the interactions of post-quantum cryptography with protocols, under the supervision of Peter Schwabe.

### Interests

- Cryptography
- Post-Quantum
- Cryptography
- Protocols
- Information Security

### S ë t) $\odot$ C

### Biography

### Education

- MSc in Computing Science, 2018 Radboud University Nijmegen
- BSc in Computing Science (Informatica), 2015 Radboud University Nijmegen
- **BSc in Information Sciences** (Informatiekunde), 2015 Radboud University Nijmegen



## 삸 Connection security for thomwiggers.nl A You are securely connected to this site.

• • •

Page Info - ht

General Media

### Website Identity

Website:	thomwiggers.nl
Owner:	This website does no

Verified by: Cloudflare, Inc.

### Privacy & History

Have I visited this website prior to t

Is this website storing information of computer?

Have I saved any passwords for this

### **Technical Details**

Versleutelde verbinding (TLS\_AES\_ De pagina die u bekijkt is versleutel verzonden.

Versleuteling maakt het moeilijk voo bekijken die tussen computers word onwaarschijnlijk dat iemand deze pa werd verzonden.

ttne://thomwiggers.nl/			
ttps://thomwiggers.ni/			
Permis	sions	Security	
ot supply ow	nership i	information. View Certificate	
today? on my	Yes, 112 No	2 times Clear Cookies and Site Data	
s website?	No	View Saved Passwords	
_128_GCM_SHA256, 128-bits sleutels, TLS 1.3) Id voordat deze over het internet werd or onbevoegde personen om gegevens te den uitgewisseld. Het is daarom agina heeft gelezen terwijl hij over het netwerk			
		2	

TLS

## Checklist Requirements for TLS

- Establish a shared secret key for application traffic
- Transmit the identity during the protocol
  - so we don't need to know it beforehand
- Be Secure

## **Transport Layer Security** A brief history

- 1995: SSL 2.0 ("Secure Sockets Layer") 🐹 (insecure)
- 1996: SSL 3.0 update 🔜 (insecure)
  - Already fixes many problems in 2.0
- 1999: TLS 1.0 (deprecated)
- 2006: TLS 1.1 (deprecated)
- 2008: TLS 1.2 (okay with the right config)
- 2018: TLS 1.3



Ser	Ver
Understand [options]	
ets do Coptions]	
rtificate + Signature	
eg exchange	
ey Exchange	
HMAC	
ncrypted DATA 7	

## **TLS 1.2 problems** AKA why you should use TLS 1.3

- Too many round-trips
- Certificates are sent in the clear
  - Everybody can see you're connecting to <u>wggrs.nl</u>
  - Especially problematic for client authentication
- A lot of legacy cryptography and patches against attacks

# Attacks

## **Attacks on TLS** A *c* very incomplete *history*

- 1998, 2006: **Bleichenbacher** breaks RSA encryption and RSA signatures using errors as side-channel
- 2011: **BEAST**: breaks SSL 3.0 and TLS 1.0 (nobody was using TLS 1.1 (2006) or 1.2 (2008)...)
  - avoid attack by using RC4 (but since 2013 RC4 is considered x...)
- 2012/2013: CRIME / BREACH: compression in TLS is bad
- 2013: Lucky Thirteen: timing attack on encrypt-then-MAC
- 2014: **POODLE**: destroys SSL 3.0
- 2014: Bleichenbacher again (**BERserk**): signature forgery
- 2015/2016: FREAK / Logjam
  - implementation flaws downgrade to EXPORT cryptography
- 2016: **DROWN**: use the server's SSLv2 support to break SSLv3/TLS 1.{0,1,2}
- 2018: **ROBOT**: Bleichenbacher's 1998 attack is still valid on many TLS 1.2 implementations

## Attacks on TLS Some common themes

- Attacks on old versions of TLS remain valid for decades
  - XP, Vista, Android <5 never supported TLS 1.1, 1.2
- Many attacks are possible because legacy algorithms are never turned off by servers
  - FREAK/Logjam: 512-bit RSA/Diffie-Hellman ('Export' crypto)
- Setting up TLS servers is a massive headache
  - So many ciphersuites, key exchange groups, ...

Description 🗉 TLS\_NULL\_WITH\_NULL\_NULL TLS\_RSA\_WITH\_NULL\_MD5 TLS\_RSA\_WITH\_NULL\_SHA TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5 TLS\_RSA\_WITH\_RC4\_128\_MD5 TLS\_RSA\_WITH\_RC4\_128\_SHA TLS\_RSA\_EXPORT\_WITH\_RC2\_CBC\_40\_MD5 TLS\_RSA\_WITH\_IDEA\_CBC\_SHA TLS\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA TLS\_RSA\_WITH\_DES\_CBC\_SHA TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA TLS\_DH\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA TLS\_DH\_DSS\_WITH\_DES\_CBC\_SHA TLS\_DH\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA TLS\_DH\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA TLS\_DH\_RSA\_WITH\_DES\_CBC\_SHA TLS\_DH\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA TLS\_DHE\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA TLS\_DHE\_DSS\_WITH\_DES\_CBC\_SHA TLS\_DHE\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA TLS\_DHE\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA TLS\_DHE\_RSA\_WITH\_DES\_CBC\_SHA TLS\_DHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA TLS\_DH\_anon\_EXPORT\_WITH\_RC4\_40\_MD5 TLS\_DH\_anon\_WITH\_RC4\_128\_MD5 TLS\_DH\_anon\_EXPORT\_WITH\_DES40\_CBC\_SHA TLS\_DH\_anon\_WITH\_DES\_CBC\_SHA TLS\_DH\_anon\_WITH\_3DES\_EDE\_CBC\_SHA Reserved to avoid conflicts with SSLv3 TLS\_KRB5\_WITH\_DES\_CBC\_SHA TLS\_KRB5\_WITH\_3DES\_EDE\_CBC\_SHA TLS\_KRB5\_WITH\_RC4\_128\_SHA TLS\_KRB5\_WITH\_IDEA\_CBC\_SHA TLS\_KRB5\_WITH\_DES\_CBC\_MD5 TLS\_KRB5\_WITH\_3DES\_EDE\_CBC\_MD5 TLS\_KRB5\_WITH\_RC4\_128\_MD5 TLS\_KRB5\_WITH\_IDEA\_CBC\_MD5 TLS\_KRB5\_EXPORT\_WITH\_DES\_CBC\_40\_SHA TLS\_KRB5\_EXPORT\_WITH\_RC2\_CBC\_40\_SHA TLS\_KRB5\_EXPORT\_WITH\_RC4\_40\_SHA TLS\_KRB5\_EXPORT\_WITH\_DES\_CBC\_40\_MD5 TLS\_KRB5\_EXPORT\_WITH\_RC2\_CBC\_40\_MD5 TLS\_KRB5\_EXPORT\_WITH\_RC4\_40\_MD5

TLS\_RSA\_WITH\_CAMELLIA\_256\_CBC\_SHA TLS\_DH\_DSS\_WITH\_CAMELLIA\_256\_CBC\_SHA TLS\_DH\_RSA\_WITH\_CAMELLIA\_256\_CBC\_SHA TLS\_DHE\_DSS\_WITH\_CAMELLIA\_256\_CBC\_SHA TLS\_DHE\_RSA\_WITH\_CAMELLIA\_256\_CBC\_SHA TLS\_DH\_anon\_WITH\_CAMELLIA\_256\_CBC\_SHA TLS\_PSK\_WITH\_RC4\_128\_SHA TLS\_PSK\_WITH\_3DES\_EDE\_CBC\_SHA TLS\_PSK\_WITH\_AES\_128\_CBC\_SHA TLS\_PSK\_WITH\_AES\_256\_CBC\_SHA TLS\_DHE\_PSK\_WITH\_RC4\_128\_SHA TLS\_DHE\_PSK\_WITH\_3DES\_EDE\_CBC\_SHA TLS\_DHE\_PSK\_WITH\_AES\_128\_CBC\_SHA TLS\_DHE\_PSK\_WITH\_AES\_256\_CBC\_SHA TLS\_RSA\_PSK\_WITH\_RC4\_128\_SHA TLS\_RSA\_PSK\_WITH\_3DES\_EDE\_CBC\_SHA TLS\_RSA\_PSK\_WITH\_AES\_128\_CBC\_SHA TLS\_RSA\_PSK\_WITH\_AES\_256\_CBC\_SHA TLS\_RSA\_WITH\_SEED\_CBC\_SHA TLS\_DH\_DSS\_WITH\_SEED\_CBC\_SHA TLS\_DH\_RSA\_WITH\_SEED\_CBC\_SHA TLS\_DHE\_DSS\_WITH\_SEED\_CBC\_SHA TLS\_DHE\_RSA\_WITH\_SEED\_CBC\_SHA TLS\_DH\_anon\_WITH\_SEED\_CBC\_SHA TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256 TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384 TLS\_DHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 TLS\_DHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384 TLS\_DH\_RSA\_WITH\_AES\_128\_GCM\_SHA256 TLS\_DH\_RSA\_WITH\_AES\_256\_GCM\_SHA384 TLS\_DHE\_DSS\_WITH\_AES\_128\_GCM\_SHA256 TLS\_DHE\_DSS\_WITH\_AES\_256\_GCM\_SHA384 TLS\_DH\_DSS\_WITH\_AES\_128\_GCM\_SHA256 TLS\_DH\_DSS\_WITH\_AES\_256\_GCM\_SHA384 TLS\_DH\_anon\_WITH\_AES\_128\_GCM\_SHA256 TLS\_DH\_anon\_WITH\_AES\_256\_GCM\_SHA384 TLS\_PSK\_WITH\_AES\_128\_GCM\_SHA256 TLS\_PSK\_WITH\_AES\_256\_GCM\_SHA384 TLS\_DHE\_PSK\_WITH\_AES\_128\_GCM\_SHA256 TLS\_DHE\_PSK\_WITH\_AES\_256\_GCM\_SHA384 TLS\_RSA\_PSK\_WITH\_AES\_128\_GCM\_SHA256 TLS\_RSA\_PSK\_WITH\_AES\_256\_GCM\_SHA384 TLS\_PSK\_WITH\_AES\_128\_CBC\_SHA256 TLS\_PSK\_WITH\_AES\_256\_CBC\_SHA384 TLS\_PSK\_WITH\_NULL\_SHA256 TLS\_PSK\_WITH\_NULL\_SHA384 TLS\_DHE\_PSK\_WITH\_AES\_128\_CBC\_SHA256 TLS\_DHE\_PSK\_WITH\_AES\_256\_CBC\_SHA384 TLS\_DHE\_PSK\_WITH\_NULL\_SHA256 TLS\_DHE\_PSK\_WITH\_NULL\_SHA384 TLS\_RSA\_PSK\_WITH\_AES\_128\_CBC\_SHA256 TLS\_RSA\_PSK\_WITH\_AES\_256\_CBC\_SHA384 TLS\_RSA\_PSK\_WITH\_NULL\_SHA256 TLS\_RSA\_PSK\_WITH\_NULL\_SHA384

TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256 TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384 TLS\_ECDH\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256 TLS\_ECDH\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384 TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256 TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384 TLS\_ECDH\_RSA\_WITH\_AES\_128\_CBC\_SHA256 TLS\_ECDH\_RSA\_WITH\_AES\_256\_CBC\_SHA384 TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256 TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384 TLS\_ECDH\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256 TLS\_ECDH\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384 TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384 TLS\_ECDH\_RSA\_WITH\_AES\_128\_GCM\_SHA256 TLS\_ECDH\_RSA\_WITH\_AES\_256\_GCM\_SHA384 TLS\_ECDHE\_PSK\_WITH\_RC4\_128\_SHA TLS\_ECDHE\_PSK\_WITH\_3DES\_EDE\_CBC\_SHA TLS\_ECDHE\_PSK\_WITH\_AES\_128\_CBC\_SHA TLS\_ECDHE\_PSK\_WITH\_AES\_256\_CBC\_SHA TLS\_ECDHE\_PSK\_WITH\_AES\_128\_CBC\_SHA256 TLS\_ECDHE\_PSK\_WITH\_AES\_256\_CBC\_SHA384 TLS\_ECDHE\_PSK\_WITH\_NULL\_SHA TLS\_ECDHE\_PSK\_WITH\_NULL\_SHA256 TLS\_ECDHE\_PSK\_WITH\_NULL\_SHA384 TLS\_RSA\_WITH\_ARIA\_128\_CBC\_SHA256 TLS\_RSA\_WITH\_ARIA\_256\_CBC\_SHA384 TLS\_DH\_DSS\_WITH\_ARIA\_128\_CBC\_SHA256 TLS\_DH\_DSS\_WITH\_ARIA\_256\_CBC\_SHA384 TLS\_DH\_RSA\_WITH\_ARIA\_128\_CBC\_SHA256 TLS\_DH\_RSA\_WITH\_ARIA\_256\_CBC\_SHA384 TLS\_DHE\_DSS\_WITH\_ARIA\_128\_CBC\_SHA256 TLS\_DHE\_DSS\_WITH\_ARIA\_256\_CBC\_SHA384 TLS\_DHE\_RSA\_WITH\_ARIA\_128\_CBC\_SHA256 TLS\_DHE\_RSA\_WITH\_ARIA\_256\_CBC\_SHA384 TLS\_DH\_anon\_WITH\_ARIA\_128\_CBC\_SHA256 TLS\_DH\_anon\_WITH\_ARIA\_256\_CBC\_SHA384 TLS\_ECDHE\_ECDSA\_WITH\_ARIA\_128\_CBC\_SHA256 TLS\_ECDHE\_ECDSA\_WITH\_ARIA\_256\_CBC\_SHA384 TLS\_ECDH\_ECDSA\_WITH\_ARIA\_128\_CBC\_SHA256 TLS\_ECDH\_ECDSA\_WITH\_ARIA\_256\_CBC\_SHA384 TLS\_ECDHE\_RSA\_WITH\_ARIA\_128\_CBC\_SHA256 TLS\_ECDHE\_RSA\_WITH\_ARIA\_256\_CBC\_SHA384 TLS\_ECDH\_RSA\_WITH\_ARIA\_128\_CBC\_SHA256 TLS\_ECDH\_RSA\_WITH\_ARIA\_256\_CBC\_SHA384 TLS\_RSA\_WITH\_ARIA\_128\_GCM\_SHA256 TLS\_RSA\_WITH\_ARIA\_256\_GCM\_SHA384 TLS\_DHE\_RSA\_WITH\_ARIA\_128\_GCM\_SHA256 TLS\_DHE\_RSA\_WITH\_ARIA\_256\_GCM\_SHA384 TLS\_DH\_RSA\_WITH\_ARIA\_128\_GCM\_SHA256 TLS\_DH\_RSA\_WITH\_ARIA\_256\_GCM\_SHA384

Ciphersuites in TLS

# This isn't even all of them!

TLS\_ECDHE\_ECDSA\_WITH\_CAMELLIA\_128\_CBC\_SHA256 TLS\_ECDHE\_ECDSA\_WITH\_CAMELLIA\_256\_CBC\_SHA384 TLS\_ECDH\_ECDSA\_WITH\_CAMELLIA\_128\_CBC\_SHA256 TLS\_ECDH\_ECDSA\_WITH\_CAMELLIA\_256\_CBC\_SHA384 TLS\_ECDHE\_RSA\_WITH\_CAMELLIA\_128\_CBC\_SHA256 TLS\_ECDHE\_RSA\_WITH\_CAMELLIA\_256\_CBC\_SHA384 TLS\_ECDH\_RSA\_WITH\_CAMELLIA\_128\_CBC\_SHA256 TLS\_ECDH\_RSA\_WITH\_CAMELLIA\_256\_CBC\_SHA384 TLS\_RSA\_WITH\_CAMELLIA\_128\_GCM\_SHA256 TLS\_RSA\_WITH\_CAMELLIA\_256\_GCM\_SHA384 TLS\_DHE\_RSA\_WITH\_CAMELLIA\_128\_GCM\_SHA256 TLS\_DHE\_RSA\_WITH\_CAMELLIA\_256\_GCM\_SHA384 TLS\_DH\_RSA\_WITH\_CAMELLIA\_128\_GCM\_SHA256 TLS DH RSA WITH CAMELLIA 256 GCM SHA384 TLS\_DHE\_DSS\_WITH\_CAMELLIA\_128\_GCM\_SHA256 TLS\_DHE\_DSS\_WITH\_CAMELLIA\_256\_GCM\_SHA384 TLS\_DH\_DSS\_WITH\_CAMELLIA\_128\_GCM\_SHA256 TLS\_DH\_DSS\_WITH\_CAMELLIA\_256\_GCM\_SHA384 TLS\_DH\_anon\_WITH\_CAMELLIA\_128\_GCM\_SHA256 TLS\_DH\_anon\_WITH\_CAMELLIA\_256\_GCM\_SHA384 TLS\_ECDHE\_ECDSA\_WITH\_CAMELLIA\_128\_GCM\_SHA256 TLS\_ECDHE\_ECDSA\_WITH\_CAMELLIA\_256\_GCM\_SHA384 TLS\_ECDH\_ECDSA\_WITH\_CAMELLIA\_128\_GCM\_SHA256 TLS\_ECDH\_ECDSA\_WITH\_CAMELLIA\_256\_GCM\_SHA384 TLS\_ECDHE\_RSA\_WITH\_CAMELLIA\_128\_GCM\_SHA256 TLS\_ECDHE\_RSA\_WITH\_CAMELLIA\_256\_GCM\_SHA384 TLS\_ECDH\_RSA\_WITH\_CAMELLIA\_128\_GCM\_SHA256 TLS\_ECDH\_RSA\_WITH\_CAMELLIA\_256\_GCM\_SHA384 TLS PSK WITH CAMELLIA 128 GCM SHA256 TLS\_PSK\_WITH\_CAMELLIA\_256\_GCM\_SHA384 TLS\_DHE\_PSK\_WITH\_CAMELLIA\_128\_GCM\_SHA256 TLS\_DHE\_PSK\_WITH\_CAMELLIA\_256\_GCM\_SHA384 TLS\_RSA\_PSK\_WITH\_CAMELLIA\_128\_GCM\_SHA256 TLS\_RSA\_PSK\_WITH\_CAMELLIA\_256\_GCM\_SHA384 TLS\_PSK\_WITH\_CAMELLIA\_128\_CBC\_SHA256 TLS\_PSK\_WITH\_CAMELLIA\_256\_CBC\_SHA384 TLS\_DHE\_PSK\_WITH\_CAMELLIA\_128\_CBC\_SHA256 TLS\_DHE\_PSK\_WITH\_CAMELLIA\_256\_CBC\_SHA384 TLS\_RSA\_PSK\_WITH\_CAMELLIA\_128\_CBC\_SHA256 TLS\_RSA\_PSK\_WITH\_CAMELLIA\_256\_CBC\_SHA384 TLS\_ECDHE\_PSK\_WITH\_CAMELLIA\_128\_CBC\_SHA256 TLS\_ECDHE\_PSK\_WITH\_CAMELLIA\_256\_CBC\_SHA384 TLS\_RSA\_WITH\_AES\_128\_CCM TLS\_RSA\_WITH\_AES\_256\_CCM TLS\_DHE\_RSA\_WITH\_AES\_128\_CCM TLS\_DHE\_RSA\_WITH\_AES\_256\_CCM TLS\_RSA\_WITH\_AES\_128\_CCM\_8 TLS\_RSA\_WITH\_AES\_256\_CCM\_8 TLS\_DHE\_RSA\_WITH\_AES\_128\_CCM\_8 TLS\_DHE\_RSA\_WITH\_AES\_256\_CCM\_8 TLS\_PSK\_WITH\_AES\_128\_CCM TLS\_PSK\_WITH\_AES\_256\_CCM TLS\_DHE\_PSK\_WITH\_AES\_128\_CCM TLS\_DHE\_PSK\_WITH\_AES\_256\_CCM TLS\_PSK\_WITH\_AES\_128\_CCM\_8 TLS\_PSK\_WITH\_AES\_256\_CCM\_8 TLS\_PSK\_DHE\_WITH\_AES\_128\_CCM\_8



## **Checklist** TLS 1.2 and earlier

- Second Secret Key for application traffic
- Transmit the identity during the protocol
  - so we don't need to know it beforehand
- Be secure
  - Be robust against attacks
  - Protect identities during the handshake

## **TLS 1.3** Wishlist

- Secure handshake
  - More privacy
  - Only forward secret key exchanges
  - Get rid of MD5, SHA1, 3DES, EXPORT, NULL, ...
- Simplify parameters
- More robust cryptography
- Faster, 1-RTT protocol
- 0-RTT resumption

## **TLS 1.3** RFC8446

- Move key exchange into the first two messages
- Encrypt as much as possible
- Be done as soon as possible

Client		Server	
Key ^ ClientHello Exch   + key_share*   + signature_a   + psk_key_exe v + pre shared	algorithms* change_modes* kev*>		
		ServerHello	^ Key
		+ key_share*	Exch
		+ pre_shared_key*	v
	+	<pre>{EncryptedExtensions}</pre>	^ Server
	+	{CertificateRequest*}	v Params
		{Certificate*}	^
		{CertificateVerify*}	Auth
		{Finished}	v
	<	[Application Data*]	
<pre>^ {Certificate Auth   {Certificate v {Finished}</pre>	⊧} /erify*} >		
[Application	Data] <>	[Application Data]	
	Client Key ^ ClientHello Exch   + key_share*   + signature_a   + psk_key_exc v + pre_shared_ v + pre_shared_	Client Key ^ ClientHello Exch   + key_share*   + signature_algorithms*   + psk_key_exchange_modes* v + pre_shared_key*> v + pre_shared_key*> Auth   {Certificate*} Auth   {CertificateVerify*} v {Finished}> [Application Data] <>	Client Server Key ^ ClientHello Exch   + key_share*   + signature_algorithms*   + psk_key_exchange_modes* v + pre_shared_key*> ServerHello + key_share* + pre_shared_key* {EncryptedExtensions} {CertificateRequest*} {CertificateRequest*} {CertificateVerify*} {Finished} [Application Data] v {Finished}> [Application Data] <> [Application Data]

## TLS 1.3 full handshake The crypto

- Key exchange via ECDH
  - only ephemeral key exchange
- Server Authentication: Signature
- Handshake authentication: HMAC-SHA256
  - "key confirmation"
- AEAD: only AES-GCM or ChaCha20-Poly1305

9			
[	Client		Serve
		TCP SYN	static (sig): pk <sub>S</sub> , sk
		TCP SYN-ACK	
	$x \leftarrow \mathbb{Z}_q$	$g^x$	
-			y ←s ℤ
			$ss \leftarrow g^x$
			$K \leftarrow KDF(s)$
	$g^y$ , AEAD <sub>K</sub> (o	ert[pk <sub>S</sub> ]  Sig(sk <sub>S</sub> , transcript)	key confirmation)
		AEAD <sub>K'</sub> (key confirmation	ı)
-		AEAD $_{K''}$ (application data	.)
-		AEAD <sub>K</sub> (application data	ı)

TLS 1.3 overview K, K', K", K": bunch of purpose-specific keys



## TLS 1.3 Resumption and 0-RTT Got a ticket?

- If you have a pre-shared key, you can do a bunch of stuff faster!
- Use PSK to compute traffic secret
- Ephemeral key exchange optional
- Use PSK to encrypt "Early Data"

ClientHello + early data + key share\* + psk key exchange modes + pre shared key (Application Data\*) \_\_\_\_> ServerHello + pre\_shared\_key + key share\* {EncryptedExtensions} + early data\* {Finished} [Application Data\*] <----(EndOfEarlyData) {Finished} [Application Data] [Application Data] <---->

## **0-RTT caveats RFC 8446 page 18**

**IMPORTANT NOTE:** The security properties for 0-RTT data are weaker than those for other kinds of TLS data. Specifically:

- 1. This data is not forward secret, as it is encrypted solely under keys derived using the offered PSK.
- 2. There are no guarantees of non-replay between connections. Protection against replay for ordinary TLS 1.3 1-RTT data is provided via the server's Random value, but 0-RTT data does not depend on the ServerHello and therefore has weaker guarantees. This is especially relevant if the data is authenticated either with TLS client authentication or inside the application protocol. The same warnings apply to any use of the early exporter master secret.

0-RTT data cannot be duplicated within a connection (i.e., the server will not process the same data twice for the same connection), and an attacker will not be able to make 0-RTT data appear to be 1-RTT data (because it is protected with different keys). Appendix E.5 contains a description of potential attacks, and Section 8 describes mechanisms which the server can use to limit the impact of replay.

## **0-RTT?** For the impatient

- Siri requests
- GET requests on websites\*
- Other stateless stuff

But are you sure that your application is completely robust against replays? GET /?query=INSERT into payments (to, amount) VALUES ("thom", 1000);

## **TLS 1.3 standardization** A brief evaluation

- Strong collaboration with academics for protocol evaluation
  - Proofs on pen/paper, and using tools like ProVerif, Tamarin
- Academic results influenced protocol design
- But TLS working group gonna TLS working group
  - State machines are still only in the appendix

## **TLS 1.3** Wishlist

Secure handshake

More privacy

Only forward secret key exchanges

**M**Less MD5

Simplify parameters

More robust cryptography

Faster, 1-RTT protocol

**0**-RTT resumption

**DPost-Quantum?** 



PK

• • •

Page Info - ht

General Media

### Website Identity

Website:	thomwiggers.nl
Owner:	This website does no

Verified by: Cloudflare, Inc.

### Privacy & History

Have I visited this website prior to t

Is this website storing information of computer?

Have I saved any passwords for this

### **Technical Details**

Versleutelde verbinding (TLS\_AES\_ De pagina die u bekijkt is versleutel verzonden.

Versleuteling maakt het moeilijk voo bekijken die tussen computers word onwaarschijnlijk dat iemand deze pa werd verzonden.

tps://thomwiggers.nl/		
Permissions Security		
t supply ownership information.		
View Certificate		
oday? Yes, 112 times n my No Clear Cookies and Site Data		
website? No View Saved Passwords		
_128_GCM_SHA256, 128-bits sleutels, TLS 1.3) Id voordat deze over het internet werd		
or onbevoegde personen om gegevens te den uitgewisseld. Het is daarom agina heeft gelezen terwijl hij over het netwerk		
2		

### Certificate

sni.cloudflaressl.com	Cloudflare Inc ECC CA-3	Baltimore CyberTrust Root
Subject Name		
Country	US	
State/Province	California	
Locality	San Francisco	
Organization	Cloudflare, Inc.	
Common Name	sni.cloudflaressl.com	
Issuer Name		
Country		
Organization	US Claudflara Inc	
Common Namo	Cloudflare Inc.	
Common Name	Cloudhare Inc ECC CA-3	
Validity		
Not Before	Wed, 16 Jun 2021 00:00:00 GMT	
Not After	Wed, 15 Jun 2022 23:59:59 GMT	
Subject Alt Names		
DNS Name	thomwiggers.nl	
DNS Name	sni.cloudflaressl.com	
DNS Name	*.thomwiggers.nl	

### Public Key Info

Algorithm	Elliptic Curve
Key Size	256
Curve	P-256
Public Value	04:04:FF:B8:9F:66:B9:D5:CE:40:91:4B:B7:B4:8C:B4:D2:C4:17:E7:AA:75:2

### Miscellaneous

Serial Number	05:E1:B4:51:22:F8:E4:1A:9F:87:F0:61:D0:40:BD:07
Signature Algorithm	ECDSA with SHA-256
Version	3
Download	PEM (cert) PEM (chain)

### Fingerprints

SHA-256	B3:D7:D5:C2:9A:ED:DE:A1:AA:7C:EA:9E:21:E9:A7:4F:6C:DA:7C:40:86:CA:8
SHA-1	8E:D8:3E:CC:C1:95:D9:25:32:E9:97:47:30:13:6D:9D:42:93:E6:83

Basic Constraints	
Certificate Authority	No
• ****	
Key Usages	
Purposes	Digital Signature
Extended Key Usages	
Purposes	Server Authentication, Client Authentication

## **Public Key Infrastructure** Oversimplified

- Certificate Authorities (CA)
- Become a trusted CA by:
  - spending 💰 💰 on audits
  - <u>convince vendors to install your certificate</u>
- Vendors trust CAs to check if I own <u>wggrs.nl</u>
- Intermediate CA certs make key management easier
  - (offline master signing key, etc)





## Aside: PKI open problems What we've oversimplified

- Certificate issuance  $\bullet$
- Certificate Revocation
  - Certificate Revocation Lists (CRL)
  - Online Certificate Status Protocol (OCSP)
- Any trusted CA can issue a certificate for anyone
  - Famously abused by Iran(?) to attack Gmail in <u>DigiNotar.nl hack</u>
  - "Certificate Transparancy" (CT)

### Authority Info (AIA)

Location	http://ocsp.digicert.com
Method	Online Certificate Status Protocol (OCSP)
Location Method	http://cacerts.digicert.com/CloudflareIncECCCA-3.crt CA Issuers

Log ID	29:79:BE:F0:9E:39:39:21:F0:56:73:9F:63:A5:77:E5:BE:57:7D:9C:60:0A:F8:
Name	Google "Argon2022"
Signature Algorithm	SHA-256 ECDSA
Version	1
Timestamp	Wed, 16 Jun 2021 17:11:33 GMT
Log ID	22:45:45:07:59:55:24:56:96:3F:A1:2F:F1:F7:6D:86:E0:23:26:63:AD:C0:4B
Name	DigiCert Yeti2022
Signature Algorithm	SHA-256 ECDSA
Version	1
Timestamp	Wed, 16 Jun 2021 17:11:33 GMT
Log ID	51:A3:B0:F5:FD:01:79:9C:56:6D:B8:37:78:8F:0C:A4:7A:CC:1B:27:CB:F7:9E
Name	DigiCert Nessie2022
Signature Algorithm	SHA-256 ECDSA
Version	1
Timestamp	Wed, 16 Jun 2021 17:11:33 GMT

# **Post-Quantum TLS**

## Post-Quantum Crypto





Information Technology Laboratory

### **COMPUTER SECURITY RESOURCE CENTER**

PROJECTS

### **Post-Quantum Cryptography** PQC

f y

### Overview

Post-Quantum Encryption: A Q&A With NIST's Matt Scholl

Post-Quantum Cryptography: the Good, the Bad, and the Powerful (video)

NIST has initiated a process to solicit, evaluate, and standardize one or more quantum-resistant public-key cryptographic algorithms. Full details can be found in the <u>Post-Quantum Cryptography Standardization</u> page.

The <u>Round 3 candidates</u> were announced July 22, 2020. <u>NISTIR 8309</u>, Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process is now available. NIST has developed <u>Guidelines for Submitting Tweaks</u> for Third Round Finalists and Candidates.

### Background

In recent years, there has been a substantial amount of research on quantum computers – machines that exploit quantum mechanical phenomena to solve mathematical problems that are difficult or intractable for conventional computers. If large-scale quantum computers are ever built, they will be able to break many of the public-key cryptosystems currently in use. This would seriously compromise the confidentiality and integrity of digital communications on the Internet and elsewhere. The goal of *post-quantum cryptography* (also called quantum-resistant cryptography) is to develop cryptographic systems that are secure against both quantum and classical computers, and can interoperate with existing communications protocols and networks.

Search CSRC Q

 $\equiv$  CSRC MENU



### ℅ PROJECT LINKS

Overview

FAQs

News & Updates

Events

Publications

Presentations

ADDITIONAL PAGES

### **Post-Quantum Cryptography Standardization**

Call for Proposals Example Files Round 1 Submissions Round 2 Submissions Round 3 Submissions Workshops and Timeline Round 3 Seminars

## **TLS 1.3 Pre-Quantum**





### Server





## **TLS 1.3 Post-Quantum!!!1!**







## **Post-Quantum KEMs Key Encapsulation Mechanisms**

Description Operation  $(pk, sk) \leftarrow \text{KEM-KeyGen}()$  Generates a public/private key pair.  $(K, ct) \leftarrow \operatorname{KEM-Encaps}(pk)$ as ct.

- Generates shared key K and encapsulates it to public key pk
- $K \leftarrow \text{KEM-Decaps}(ct, sk)$  Decapsulates ct using sk to obtain K



## **Post-Quantum key sizes** New tradeoffs in cryptography

Signature scheme		Public key (bytes)	Signature (bytes)
RSA-2048	Factoring	272	256
Elliptic curves	Elliptic curve discrete logarithm	32	32
Dilithium	Lattice-based (MLWE/MSIS)	1,184	2,044
Falcon	Lattice-based (NTRU)	897	690
XMSS	Hash-based	32	979
GeMSS	Multi-variate	352,180	32
KEM		Public key (bytes)	Ciphertext (bytes)
RSA-2048	Factoring	272	256
Elliptic curves	Elliptic curve discrete logarithm	32	32
Kyber	Lattice-based (MLWE)	800	768
NTRU	Lattice-based (NTRU)	699	699
Saber	Lattice-based (MLWR)	672	736
SIKE	Isogeny-based	330	330
SIKE compressed	Isogeny-based	197	197
Classic McEliece	Code-based	261,120	128

# PQ signatures are big and/or slow and/or need hw support

### Use key exchange for authentication



## Authentication

### **Explicit authentication:**

Alice receives assurance that she really is talking to Bob

- Signed Diffie-Hellman
- SIGMA
- TLS 1.3

## Implicit authentication:

Alice is assured that only Bob would be able to compute the shared secret

- Signal
- Wireguard
- Noise Framework

Can always use MAC to confirm key

## TLS handshake authentication Recap

• Signatures allow us to authenticate immediately!

Client Server \_\_\_\_> ClientHello ServerHello <-----<...> <CertificateRequest> <Certificate> <CertificateVerify> <Finished> <-----<Certificate> <CertificateVerify> <Finished> \_\_\_\_> [Application Data] <----> [Application Data] <msg>: enc. w/ keys derived from ephemeral KEX (HS) [msg]: enc. w/ keys derived from HS (MS)

## Authenticated Key Exchange via KEM An oversimplified protocol



Note that this protocol assumes that we have already exchanged the public keys!



- - ct

 $ss \leftarrow KEM.Decapsulate(ct, sk_{Douglas})$  $K \leftarrow KDF(ss)$  $MAC_{K}(\cdots)$ 

## **TLS authentication via KEM** Naively

- Signatures allow us to authenticate immediately!
- KEMs require interactivity

 Exercise for the reader: see how Diffie—Hellman's non-interactive key exchange property would have allowed us to do this more efficiently (See OPTLS by Krawczyk and Wee)

Client	Server
ClientHello	>
	< ServerHello
	<>
	<certificaterequest></certificaterequest>
	<pre>&lt; <certificate></certificate></pre>
<kemencapsulation></kemencapsulation>	>
	<pre>&lt; <finished></finished></pre>
<certificate></certificate>	>
	<pre>&lt; <kemencapsulation></kemencapsulation></pre>
<finished></finished>	>
[Application Data]	<> [Application Data]
<msg>: enc. w/ keys [msg]: enc. w/ keys</msg>	derived from ephemeral KEX (HS) derived from HS (MS)

## **KEMTLS** ACM CCS 2020

### KEM for ephemeral key exchange

### KEM for server-to-client authenticated key exchange

**Combine shared secrets** 





## **KEMTLS** The clever bit

- What can a server send to a client, before the client has said what they wanted?
- server before receiving Server's Finished message
- Avoid 2-RTT protocol
- Client can send HTTP request in same place as in TLS 1.3

• Use *implicitly authenticated* key to encrypt application message (request) to

## KEMTLS **Sizes of instantiations**

		Abbrv.	KEX (pk+ct)	Excludin HS auth (ct/sig)	g intermediate Leaf crt. subject (pk)	e CA certificate Leaf crt. (signature)	Sum excl. int. CA cert.	Including i   Int. CA crt.   subject (pk)	ntermediate C Int. CA crt. (signature)	A certificate Sum incl. int. CA crt.	Root CA (pk)	Sum TCP pay- loads of TLS HS (incl. int. CA crt.)
l KEX)	TLS 1.3	errr	ECDH (X25519) 64	RSA-2048 256	RSA-2048 272	RSA-2048 256	848	RSA-2048 272	RSA-2048 256	1376	RSA-2048 272	2829
	Min. incl. int. CA cert.	SFXR	SIKE 433	Falcon 690	Falcon 897	XMSS <sup>MT</sup> 979	2999	XMSS <sup>MT</sup> 32	Rainbow 66	3097	Rainbow 161600	5378
(Signed	Min. excl. int. CA cert.	SFRR	SIKE 433	Falcon 690	Falcon 897	Rainbow 66	2086	Rainbow 60192	Rainbow 66	62344	Rainbow 60192	64693
TLS 1.3	Assumption: MLWE+MSIS	KDDD	Kyber 1568	Dilithium 2420	Dilithium 1312	Dilithium 2420	7720	Dilithium 1312	Dilithium 2420	11452	Dilithium 1312	12639
	Assumption: NTRU	NFFF	NTRU 1398	Falcon 690	Falcon 897	Falcon 690	3675	Falcon 897	Falcon 690	5262	Falcon 897	6524
KEMTLS	Min. incl. int. CA cert.	SSXR	SIKE 433	SIKE 236	SIKE 197	XMSS <sup>MT</sup> 979	1845	XMSS <sup>MT</sup> 32	Rainbow 66	1943	Rainbow 60192	4252
	Min. excl. int. CA cert.	SSRR	SIKE 433	SIKE 236	SIKE 197	Rainbow 66	932	Rainbow 60192	Rainbow 66	61190	Rainbow 60192	63568
	Assumption: MLWE+MSIS	KKDD	Kyber 1568	Kyber 768	Kyber 800	Dilithium 2420	5556	Dilithium 1312	Dilithium 2420	9288	Dilithium 1312	10471
	Assumption: NTRU	NNFF	NTRU 1398	NTRU 699	NTRU 699	Falcon 690	3486	Falcon 897	Falcon 690	5073	Falcon 897	6359

### Table 3: Instantiations of handshakes with sizes in bytes of transmitted public-key cryptography objects (NIST round 3).



Rustls client/server with some AVX2 implementations. Emulated network: latency 31.1 ms, bandwidth 1000 Mbps, 0% packet loss. Average of 100000 iterations.

24

## **KEMTLS** Client Authentication

- Unfortunately, no nice tricks exist for the client certificate ...
- Full extra round-trip in KEMTLS

 Also: we need an extra "authenticated" handshake traffic secret to protect the client certificate

	Client	Server
	ClientHello	>
		< ServerHello
		<>
		<certificaterequest></certificaterequest>
		< <certificate> ^</certificate>
	<kemencapsulation></kemencapsulation>	
	<pre>^ {Certificate}</pre>	>
Auth		
		<pre>&lt; {KEMEncapsulation}</pre>
	{Finished}	>
	[Application Data]	>
	V	< {Finished} v
	[Application Data]	<> [Application Data]
	<msg>: enc. w/ keys</msg>	derived from ephemeral KEX (HS)
	{msg}: enc. w/ keys	derived from HS+srv. KEM Auth (AHS
	[msq]: enc. w/ keys	derived from AHS+cl. KEM Auth (MS)



## **KEMTLS-PDK Pre-Distributed Keys**

- The client often knows the server:
  - minutes

  - checking firmware updates from the same server every day

The client reasonably might know the server's long-term key

• It's the 10th time you refreshed the front page of Reddit in the past 5

You've been doom-scrolling /r/wallstreetbets is for two hours already

Or the client is a too-cheap IoT security camera spying on you for China

## **KEMTLS-PDK Pre-Distributed Keys**

- Use server's long-term (certificate) public key to encaps before ClientHello
- Send the ciphertext with ClientHello
- Don't transmit certificates anymore
- Save even more bytes

```
Client
                                         Server
ct <- KEM.Encaps(pkS)</pre>
ClientHello
 + ...
 + KemEncapsulation ---->
                                      ServerHello
                    <----
                                            <...>
                                       <Finished>
                    <----
                    <---- [Application Data]</pre>
<Finished>
                    ____>
 [Application Data] <----> [Application Data]
<msg>: enc. w/ keys derived from KEX+srv. KEM auth (HS)
```

```
[msg]: enc. w/ traffic keys derived from HS (MS)
```



## **KEMTLS-PDK** Client Authentication

- We now have an implicitly authenticated key already before we sent the ClientHello message!
- Use this to also encrypt and send over the client's certificate
- Or 0-RTT?

- No replay protection
- No forward secrecy

Client		Server
ClientHello + KemEncapsulation		
{Certificate}	>	
	<	ServerHello
		<>
		<kemencapsulation></kemencapsulation>
	<	<finished></finished>
	<	[Application Data]
<finished></finished>	>	
[Application Data]	<>	[Application Data]
{msg}: enc. w/ keys o	derived from	m srv. KEM auth (ES)
<msg>: enc. w/ keys o</msg>	derived from	m KEX+srv. KEM auth (
[msg]: enc. w/ keys o	derived from	m HS+cl. KEM auth (MS

### (HS) S)

## **KEMTLS** Benefits

- Size-optimized KEMTLS requires < ½ communication of size-optimized PQ signed-KEM</li>
- Speed-optimized KEMTLS uses 90% fewer server CPU cycles and still reduces communication
- NTRU KEX (27  $\mu$ s) 10x faster than Falcon signing (254  $\mu$ s)
- No extra round trips required until client starts sending application data
- Smaller trusted code base (no signature generation on client/server)

## **TLS ecosystem challenges** So much going on...

- Datagram TLS
- Use of TLS handshake in other protocols
  - e.g. QUIC
- Application-specific behaviour
  - e.g. HTTP3 SETTINGS frame not server authenticated
- PKI involving KEM public keys
- Long tail of implementations

## **KEMTLS** Standardizing?

- Authentication bits from KEMTLS have been submitted to the TLS working group at the Internet Engineering Task Force (IETF) (aka the RFC people)
  - <u>https://datatracker.ietf.org/doc/draft-celi-wiggers-tls-authkem/</u>
  - <u>https://wggrs.nl/docs/authkem-abridged/</u>

## **Transitioning to PQ** Wrap-up

- The transition to post-quantum means:
  - KEMs are less flexible than Diffie—Hellman
    - No non-interactive key exchange
  - PQ is bigger than ECC we got used to
  - Post-Quantum Signatures are big
- KEMTLS really explores new tradeoffs
  - Signing and key exchange are not the same operations anymore

## Thanks for your attention

• Transitioning to PQ is an opportunity to reconsider some established protocols!

