Modern Key Exchange Security Models

Benjamin Dowling

Joint work with: Florian Bergsma, Marc Fischlin, Felix Günther, Florian Kohlar, Jörg Schwenk, Douglas Stebila

Overview

- Multi-Stage Key Exchange
  - Overview
  - Adversarial Model
  - Security Properties
  - Hybrid Argument
  - Composition
- Authenticated and Confidential Channel Establishment (ACCE)
  - ACCE
  - Multi-Ciphersuite ACCE
  - Negotiable ACCE

Introduction

PhD student from Queensland University Technology
Supervisors: Assistant Prof. Douglas Stebila and Prof. Colin Boyd
This talk features work from:

A cryptographic analysis of the TLS 1.3 draft-10 Full and Pre-shared Key Handshake Protocol.
Benjamin Dowling, Felix Günther, Marc Fischlin, and Douglas Stebila. Presented at TLS 1.3 Ready or Not Workshop (TRON 2016)

A cryptographic analysis of the TLS 1.3 handshake protocol candidates.
Benjamin Dowling, Felix Günther, Marc Fischlin, and Douglas Stebila. *In Proc. of* 22nd ACM Conference on Computer and Communications Security (CCS 2015)

Modelling ciphersuite and version negotiation in the TLS protocol.
Benjamin Dowling, and Douglas Stebila. *In Proc. of* 20th Australasian Conference on Information Security and Privacy (ACISP 2015)

Multi-ciphersuite security of the Secure Shell (SSH) protocol.
Florian Bergsma, Benjamin Dowling, Florian Kohlar, Jörg Schwenk and Douglas Stebila. *In Proc. of* 21st ACM Conference on Computer and Communications Security (CCS 2014)

Multi-Stage Key Exchange Protocols

Overview

- Follows the Bellare-Rogaway AKE paradigm

Overview

- Follows the Bellare-Rogaway AKE paradigm
- Multi-Stage = outputs multiple keys that can be used in the protocol

Examples

Client $\mathcal{C}$		Server $\mathcal{S}$
server's static public key $pk_{\mathcal{S}}$, $[\mathsf{nonce}_{\mathcal{S}}]$		server's static secret key $sk_{\mathcal{S}}$
generate ephemeral keys $\mathsf{esk}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}$		
generate $\mathsf{nonce}_{\mathcal{C}}$	$\xrightarrow{\mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{aux}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}}$	
$D_1 = \mathrm{DH}(\mathsf{esk}_{\mathcal{C}}, pk_{\mathcal{S}})$		$D_1 = \mathrm{DH}(\mathsf{epk}_{\mathcal{C}}, sk_{\mathcal{S}})$
$\mathsf{PRK}_1 = \mathrm{KDF}_{\mathrm{ext}}(D_1, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$		$\mathsf{PRK}_1 = \mathrm{KDF}_{\mathrm{ext}}(D_1, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$
$\mathsf{K}_1 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_1, \mathsf{info}_1)$		$\mathsf{K}_1 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_1, \mathsf{info}_1)$
	$\xleftarrow{\{\mathsf{aux}_{\mathcal{S}}, \mathsf{tpk}_{\mathcal{S}}\}_{\mathsf{K}_1}}$	use temporary keys $\mathsf{tsk}_{\mathcal{S}}, \mathsf{tpk}_{\mathcal{S}}$
$D_2 = \mathrm{DH}(\mathsf{esk}_{\mathcal{C}}, \mathsf{tpk}_{\mathcal{S}})$		$D_2 = \mathrm{DH}(\mathsf{epk}_{\mathcal{C}}, \mathsf{tsk}_{\mathcal{S}})$
$\mathsf{PRK}_2 = \mathrm{KDF}_{\mathrm{ext}}(D_2, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$		$\mathsf{PRK}_2 = \mathrm{KDF}_{\mathrm{ext}}(D_2, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$
$\mathsf{K}_2 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_2, \mathsf{info}_2)$		$\mathsf{K}_2 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_2, \mathsf{info}_2)$

Figure 3: Expanded description of protocol run of Google's QUIC with 0-RTT handshake.

Client		Server
ClientHello: $r_c \leftarrow_{\$} \{0,1\}^{128}$ + ClientKeyShare: $X \leftarrow g^x$	$\longrightarrow$	
		ServerHello: $r_s \leftarrow_{\$} \{0,1\}^{128}$ + ServerKeyShare: $Y \leftarrow g^y$
	$\longleftarrow$	
	$H_1 \leftarrow \mathsf{H}(\mathtt{CH}\|\mathtt{SH})$ (incl. CKS & SKS)	
$\mathrm{ES} \leftarrow Y^x$		$\mathrm{ES} \leftarrow X^y$
	$\mathrm{xES} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{ES})$	
	$tk_{hs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_1 \| H_1)$ stage 1	
		{EncryptedExtensions}
		{ServerConfiguration*}
		{ServerCertificate*}: pk_S
		{CertificateRequest*}
	$H_2 \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CR}^*)$	
		{ServerCertificateVerify*}:
		$\mathtt{SCV} \leftarrow \mathsf{Sign}(sk_S, H_2)$
$\mathrm{SS} \leftarrow Y^x$		$\mathrm{SS} \leftarrow X^y$
	$\mathrm{xSS} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{SS})$	
	$H_3 \leftarrow \mathsf{H}(\mathtt{CH}\| \cdots \|\mathtt{SCV}^*)$	
	$\mathrm{FS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_2 \| H_3)$	
		{ServerFinished}:
		$\mathtt{SF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3 \| H_3)$
	$\longleftarrow$	
check $\mathsf{Verify}(pk_S, H_2, \mathtt{SCV}) = 1$		
check $\mathtt{SF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3 \| H_3)$		
{ClientCertificate*}: pk_C		
	$H_4 \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CCRT}^*)$	
{ClientCertificateVerify*}:		
$\mathtt{CCV} \leftarrow \mathsf{Sign}(sk_C, H_4)$		
	$H_{\mathrm{sess}} \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CCV}^*)$	
{ClientFinished}:		
$\mathtt{CF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4 \| H_{\mathrm{sess}})$	$\longrightarrow$	
		check $\mathsf{Verify}(pk_C, H_4, \mathtt{CCV}) = 1$
		check $\mathtt{CF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4 \| H_{\mathrm{sess}})$
	$\mathrm{mES} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_5 \| H_3)$	
	$\mathrm{mSS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_6 \| H_3)$	
	$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{mSS}, \mathrm{mES})$	
	$tk_{app} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_7 \| H_{\mathrm{sess}})$ stage 2	
		[NewSessionTicket$^{\Delta}$]: psk_id
	$\mathrm{RMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_8 \| H_{\mathrm{sess}})$ stage 3	
	$\mathrm{EMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_9 \| H_{\mathrm{sess}})$ stage 4	
record layer (application data), using AEAD with key tk_{app}		
$\longleftrightarrow$		

Examples

Client $\mathcal{C}$		Server $\mathcal{S}$
server's static public key $pk_{\mathcal{S}}$, $[\mathsf{nonce}_{\mathcal{S}}]$		server's static secret key $sk_{\mathcal{S}}$
generate ephemeral keys $\mathsf{esk}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}$		
generate $\mathsf{nonce}_{\mathcal{C}}$	$\xrightarrow{\mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{aux}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}}$	
$D_1 = \mathrm{DH}(\mathsf{esk}_{\mathcal{C}}, pk_{\mathcal{S}})$		$D_1 = \mathrm{DH}(\mathsf{epk}_{\mathcal{C}}, sk_{\mathcal{S}})$
$\mathsf{PRK}_1 = \mathrm{KDF}_{\mathrm{ext}}(D_1, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$		$\mathsf{PRK}_1 = \mathrm{KDF}_{\mathrm{ext}}(D_1, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$
$\mathsf{K}_1 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_1, \mathsf{info}_1)$		$\mathsf{K}_1 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_1, \mathsf{info}_1)$
	$\xleftarrow{\{\mathsf{aux}_{\mathcal{S}}, \mathsf{tpk}_{\mathcal{S}}\}_{\mathsf{K}_1}}$	use temporary keys $\mathsf{tsk}_{\mathcal{S}}, \mathsf{tpk}_{\mathcal{S}}$
$D_2 = \mathrm{DH}(\mathsf{esk}_{\mathcal{C}}, \mathsf{tpk}_{\mathcal{S}})$		$D_2 = \mathrm{DH}(\mathsf{epk}_{\mathcal{C}}, \mathsf{tsk}_{\mathcal{S}})$
$\mathsf{PRK}_2 = \mathrm{KDF}_{\mathrm{ext}}(D_2, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$		$\mathsf{PRK}_2 = \mathrm{KDF}_{\mathrm{ext}}(D_2, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$
$\mathsf{K}_2 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_2, \mathsf{info}_2)$		$\mathsf{K}_2 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_2, \mathsf{info}_2)$

Figure 3: Expanded description of protocol run of Google's QUIC with 0-RTT handshake.

Client		Server
ClientHello: $r_c \leftarrow_{\$} \{0,1\}^{128}$ + ClientKeyShare: $X \leftarrow g^x$	$\longrightarrow$	
	$\longleftarrow$	ServerHello: $r_s \leftarrow_{\$} \{0,1\}^{128}$ + ServerKeyShare: $Y \leftarrow g^y$
$H_1 \leftarrow \mathsf{H}(\mathtt{CH}\|\mathtt{SH})$ (incl. CKS & SKS)		
$\mathrm{ES} \leftarrow Y^x$		$\mathrm{ES} \leftarrow X^y$
$\mathrm{xES} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{ES})$		
$tk_{hs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_1 \| H_1)$		stage 1
		{EncryptedExtensions}
		{ServerConfiguration*}
		{ServerCertificate*}: pk_S
		{CertificateRequest*}
$H_2 \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CR}^*)$		
		{ServerCertificateVerify*}:
		$\mathtt{SCV} \leftarrow \mathsf{Sign}(sk_S, H_2)$
$\mathrm{SS} \leftarrow Y^x$		$\mathrm{SS} \leftarrow X^y$
$\mathrm{xSS} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{SS})$		
$H_3 \leftarrow \mathsf{H}(\mathtt{CH}\| \cdots \|\mathtt{SCV}^*)$		
$\mathrm{FS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_2 \| H_3)$		
		{ServerFinished}:
		$\mathtt{SF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3 \| H_3)$
check $\mathsf{Verify}(pk_S, H_2, \mathtt{SCV}) = 1$		
check $\mathtt{SF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3 \| H_3)$		
{ClientCertificate*}: pk_C		
$H_4 \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CCRT}^*)$		
{ClientCertificateVerify*}:		
$\mathtt{CCV} \leftarrow \mathsf{Sign}(sk_C, H_4)$		
$H_{\mathrm{sess}} \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CCV}^*)$		
{ClientFinished}:		
$\mathtt{CF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4 \| H_{\mathrm{sess}})$	$\longrightarrow$	
		check $\mathsf{Verify}(pk_C, H_4, \mathtt{CCV}) = 1$
		check $\mathtt{CF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4 \| H_{\mathrm{sess}})$
$\mathrm{mES} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_5 \| H_3)$		
$\mathrm{mSS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_6 \| H_3)$		
$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{mSS}, \mathrm{mES})$		
$tk_{app} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_7 \| H_{\mathrm{sess}})$		stage 2
		[NewSessionTicket$^{\Delta}$]: psk_id
$\mathrm{RMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_8 \| H_{\mathrm{sess}})$		stage 3
$\mathrm{EMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_9 \| H_{\mathrm{sess}})$		stage 4
record layer (application data), using AEAD with key tk_{app}	$\longleftrightarrow$	

Overview

- Follows the Bellare-Rogaway AKE paradigm
- Multi-Stage = outputs multiple keys that can be used in the protocol
- Allows post-specified session partnering

Examples

Client $\mathcal{C}$		Server $\mathcal{S}$
server's static public key $pk_{\mathcal{S}}$, $[\mathsf{nonce}_{\mathcal{S}}]$		server's static secret key $sk_{\mathcal{S}}$
generate ephemeral keys $\mathsf{esk}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}$		
generate $\mathsf{nonce}_{\mathcal{C}}$	$\xrightarrow{\mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{aux}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}}$	
$D_1 = \mathrm{DH}(\mathsf{esk}_{\mathcal{C}}, pk_{\mathcal{S}})$		$D_1 = \mathrm{DH}(\mathsf{epk}_{\mathcal{C}}, sk_{\mathcal{S}})$
$\mathsf{PRK}_1 = \mathrm{KDF}_{\mathrm{ext}}(D_1, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$		$\mathsf{PRK}_1 = \mathrm{KDF}_{\mathrm{ext}}(D_1, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$
$\mathsf{K}_1 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_1, \mathsf{info}_1)$		$\mathsf{K}_1 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_1, \mathsf{info}_1)$
	$\xleftarrow{\{\mathsf{aux}_{\mathcal{S}}, \mathsf{tpk}_{\mathcal{S}}\}_{\mathsf{K}_1}}$	use temporary keys $\mathsf{tsk}_{\mathcal{S}}, \mathsf{tpk}_{\mathcal{S}}$
$D_2 = \mathrm{DH}(\mathsf{esk}_{\mathcal{C}}, \mathsf{tpk}_{\mathcal{S}})$		$D_2 = \mathrm{DH}(\mathsf{epk}_{\mathcal{C}}, \mathsf{tsk}_{\mathcal{S}})$
$\mathsf{PRK}_2 = \mathrm{KDF}_{\mathrm{ext}}(D_2, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$		$\mathsf{PRK}_2 = \mathrm{KDF}_{\mathrm{ext}}(D_2, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$
$\mathsf{K}_2 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_2, \mathsf{info}_2)$		$\mathsf{K}_2 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_2, \mathsf{info}_2)$

Figure 3: Expanded description of protocol run of Google's QUIC with 0-RTT handshake.

Client		Server
ClientHello: $r_c \leftarrow\$ \{0,1\}^{128}$ + ClientKeyShare: $X \leftarrow g^x$	$\longrightarrow$	
		ServerHello: $r_s \leftarrow\$ \{0,1\}^{128}$ + ServerKeyShare: $Y \leftarrow g^y$
	$\longleftarrow$	
		$H_1 \leftarrow \mathsf{H}(\mathtt{CH}\|\mathtt{SH})$ (incl. CKS & SKS)
$\mathrm{ES} \leftarrow Y^x$		$\mathrm{ES} \leftarrow X^y$
		$\mathrm{xES} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{ES})$
		$tk_{hs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_1 \| H_1)$ stage 1
		{EncryptedExtensions}
		{ServerConfiguration*}
		{ServerCertificate*}: pk_S
		{CertificateRequest*}
		$H_2 \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CR}^*)$
		{ServerCertificateVerify*}:
		$\mathtt{SCV} \leftarrow \mathsf{Sign}(sk_S, H_2)$
$\mathrm{SS} \leftarrow Y^x$		$\mathrm{SS} \leftarrow X^y$
		$\mathrm{xSS} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{SS})$
		$H_3 \leftarrow \mathsf{H}(\mathtt{CH}\| \cdots \|\mathtt{SCV}^*)$
		$\mathrm{FS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_2 \| H_3)$
		{ServerFinished}:
		$\mathtt{SF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3 \| H_3)$
	$\longleftarrow$	
check $\mathsf{Verify}(pk_S, H_2, \mathtt{SCV}) = 1$		
check $\mathtt{SF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3 \| H_3)$		
{ClientCertificate*}: pk_C		
		$H_4 \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CCRT}^*)$
{ClientCertificateVerify*}:		
$\mathtt{CCV} \leftarrow \mathsf{Sign}(sk_C, H_4)$		
		$H_{\mathrm{sess}} \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CCV}^*)$
{ClientFinished}:		
$\mathtt{CF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4 \| H_{\mathrm{sess}})$	$\longrightarrow$	
		check $\mathsf{Verify}(pk_C, H_4, \mathtt{CCV}) = 1$
		check $\mathtt{CF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4 \| H_{\mathrm{sess}})$
		$\mathrm{mES} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_5 \| H_3)$
		$\mathrm{mSS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_6 \| H_3)$
		$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{mSS}, \mathrm{mES})$
		$tk_{app} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_7 \| H_{\mathrm{sess}})$ stage 2
		[NewSessionTicket$^{\Delta}$]: psk_id
		$\mathrm{RMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_8 \| H_{\mathrm{sess}})$ stage 3
		$\mathrm{EMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_9 \| H_{\mathrm{sess}})$ stage 4
record layer (application data), using AEAD with key tk_{app}		
	$\longleftrightarrow$	

Overview

- Follows the Bellare-Rogaway AKE paradigm
- Multi-Stage = outputs multiple keys that can be used in the protocol
- Allows post-specified session partnering
- Considers multiple levels of authentication
  - Key-indistinguishability of unauthenticated sessions if honestly contributed
  - Key-indistinguishability of authenticated sessions as per usual
  - Key-indistinguishability of unilaterally-authenticated sessions if honest client

Examples

Client $\mathcal{C}$		Server $\mathcal{S}$
server's static public key $pk_{\mathcal{S}}$, $[\mathsf{nonce}_{\mathcal{S}}]$		server's static secret key $sk_{\mathcal{S}}$
generate ephemeral keys $\mathsf{esk}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}$		
generate $\mathsf{nonce}_{\mathcal{C}}$	$\xrightarrow{\mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{aux}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}}$	
$D_1 = \mathrm{DH}(\mathsf{esk}_{\mathcal{C}}, pk_{\mathcal{S}})$		$D_1 = \mathrm{DH}(\mathsf{epk}_{\mathcal{C}}, sk_{\mathcal{S}})$
$\mathsf{PRK}_1 = \mathrm{KDF}_{\mathrm{ext}}(D_1, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$		$\mathsf{PRK}_1 = \mathrm{KDF}_{\mathrm{ext}}(D_1, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$
$\mathsf{K}_1 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_1, \mathsf{info}_1)$		$\mathsf{K}_1 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_1, \mathsf{info}_1)$
	$\xleftarrow{\{\mathsf{aux}_{\mathcal{S}}, \mathsf{tpk}_{\mathcal{S}}\}_{\mathsf{K}_1}}$	use temporary keys $\mathsf{tsk}_{\mathcal{S}}, \mathsf{tpk}_{\mathcal{S}}$
$D_2 = \mathrm{DH}(\mathsf{esk}_{\mathcal{C}}, \mathsf{tpk}_{\mathcal{S}})$		$D_2 = \mathrm{DH}(\mathsf{epk}_{\mathcal{C}}, \mathsf{tsk}_{\mathcal{S}})$
$\mathsf{PRK}_2 = \mathrm{KDF}_{\mathrm{ext}}(D_2, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$		$\mathsf{PRK}_2 = \mathrm{KDF}_{\mathrm{ext}}(D_2, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$
$\mathsf{K}_2 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_2, \mathsf{info}_2)$		$\mathsf{K}_2 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_2, \mathsf{info}_2)$

Figure 3: Expanded description of protocol run of Google's QUIC with 0-RTT handshake.

Client		Server
ClientHello: $r_c \leftarrow_{\$} \{0,1\}^{128}$ + ClientKeyShare: $X \leftarrow g^x$	$\longrightarrow$	
		ServerHello: $r_s \leftarrow_{\$} \{0,1\}^{128}$ + ServerKeyShare: $Y \leftarrow g^y$
	$\longleftarrow$	
		$H_1 \leftarrow \mathsf{H}(\mathtt{CH}\|\mathtt{SH})$ (incl. CKS & SKS)
$\mathrm{ES} \leftarrow Y^x$		$\mathrm{ES} \leftarrow X^y$
		$\mathrm{xES} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{ES})$
		$tk_{hs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_1 \| H_1)$ stage 1
		{EncryptedExtensions}
		{ServerConfiguration*}
		{ServerCertificate*}: pk_S
		{CertificateRequest*}
		$H_2 \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CR}^*)$
		{ServerCertificateVerify*}:
		$\mathtt{SCV} \leftarrow \mathsf{Sign}(sk_S, H_2)$
$\mathrm{SS} \leftarrow Y^x$		$\mathrm{SS} \leftarrow X^y$
		$\mathrm{xSS} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{SS})$
		$H_3 \leftarrow \mathsf{H}(\mathtt{CH}\| \cdots \|\mathtt{SCV}^*)$
		$\mathrm{FS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_2 \| H_3)$
		{ServerFinished}:
		$\mathtt{SF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3 \| H_3)$
	$\longleftarrow$	
check $\mathsf{Verify}(pk_S, H_2, \mathtt{SCV}) = 1$		
check $\mathtt{SF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3 \| H_3)$		
{ClientCertificate*}: pk_C		
		$H_4 \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CCRT}^*)$
{ClientCertificateVerify*}:		
$\mathtt{CCV} \leftarrow \mathsf{Sign}(sk_C, H_4)$		
		$H_{\mathrm{sess}} \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CCV}^*)$
{ClientFinished}:		
$\mathtt{CF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4 \| H_{\mathrm{sess}})$	$\longrightarrow$	
		check $\mathsf{Verify}(pk_C, H_4, \mathtt{CCV}) = 1$
		check $\mathtt{CF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4 \| H_{\mathrm{sess}})$
		$\mathrm{mES} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_5 \| H_3)$
		$\mathrm{mSS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_6 \| H_3)$
		$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{mSS}, \mathrm{mES})$
		$tk_{app} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_7 \| H_{\mathrm{sess}})$ stage 2
		[NewSessionTicket$^{\Delta}$]: psk_id
		$\mathrm{RMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_8 \| H_{\mathrm{sess}})$ stage 3
		$\mathrm{EMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_9 \| H_{\mathrm{sess}})$ stage 4
record layer (application data), using AEAD with key tk_{app}		
	$\longleftrightarrow$	

Secret Compromise Paradigm

- Adversary can compromise long-term keys
  - For modelling forward secrecy and leakage of long-lived secrets
- Adversary can compromise session keys
  - For modelling leakage of keys in future usage
- Adversary <u>cannot</u> compromise semi-ephemeral secrets
  - Security of TLS handshake (except 0-RTT) independent of these secrets
- Adversary <u>cannot</u> compromise ephemeral secrets or session state
  - TLS isn't intended to be secure in these cases

Adversary Model

- Adversary has full control of the network
- Adversarial Interaction
  - **NewTempKey**: creates a new temporary key pair for a party
  - **NewSession**: creates a new session with adversarial-specified details
  - **Send**: Standard send message query
  - **Corrupt:** Standard long-term key compromise
  - **Reveal:** Standard session-key compromise
  - **Test:** Plays the RoR key-indistinguishability game

Adversarial Interaction

- **Send**: If at some point the session accepts, the game pauses and:
  - Check whether the partner session has been **Revealed** or **Tested**
  - If the key is safe, the adversary can **Test** the key, getting back RoR key
  - If the adversary does not **Test**, a **Send** message tells the session to continue
  - Why? Need to ensure that the key can be used later in the protocol without allowing the adversary to trivially break the game

- **Test**: If adversary successfully Tests a key:
  - Check if honest partner exists (if unauthenticated) and if not, lose the game
  - If adversary attempts to **Test** a key that has been used, lose the game
  - If partner exists (and has been **Tested**) ensure consistency
  - Use output from **Tested** key in subsequent stages if necessary

Adversarial Interaction II

- **Reveal**: Gives the adversary the honestly generated key
  - Check whether partner session exists, and if so consider it **Revealed**
  - If key-independence unsatisfied, consider future stages **Revealed**

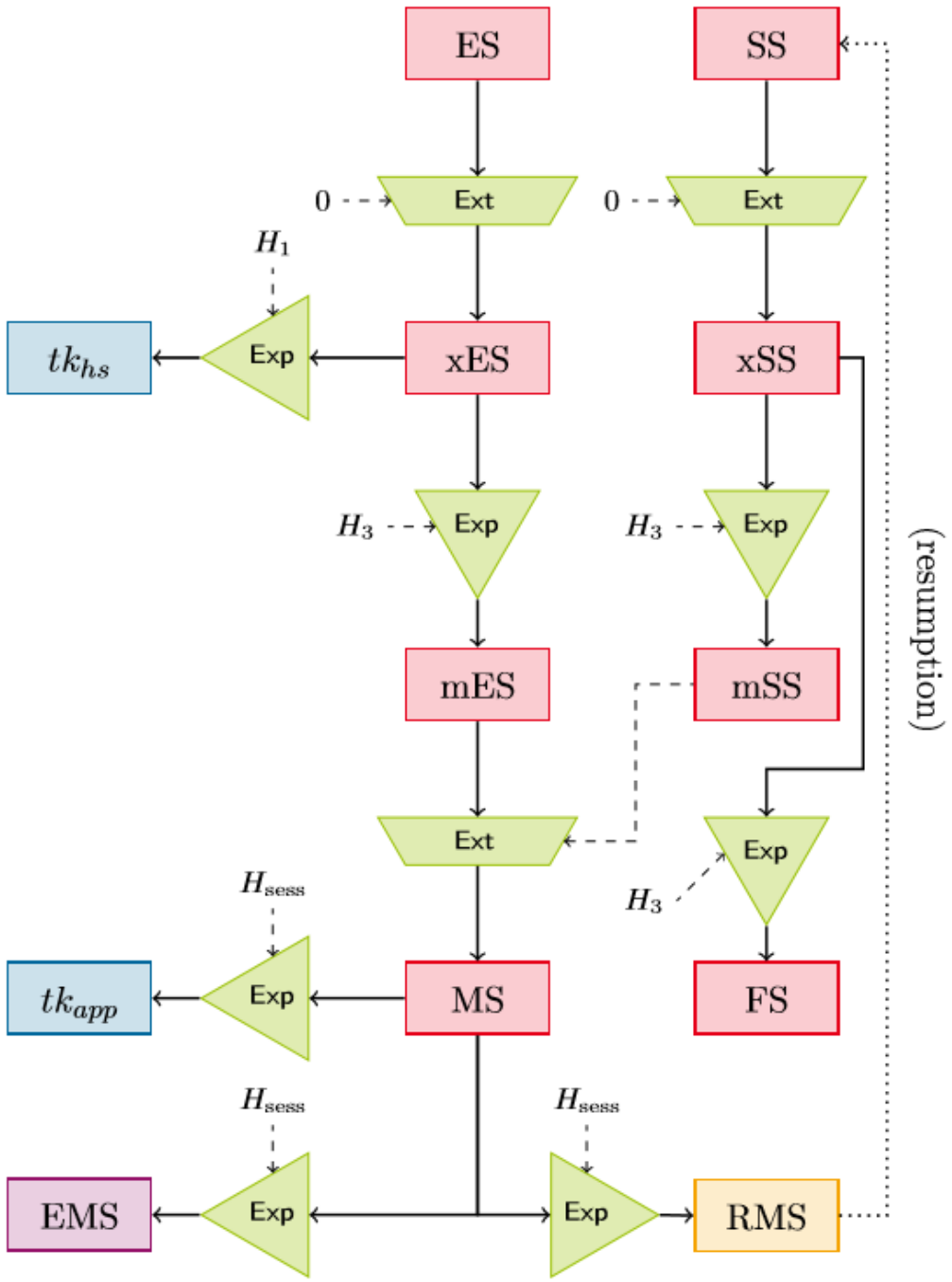
- **Corrupt**: Gives the adversary the party's long-term secret key
  - No future queries to the party allowed
  - If key-independence unsatisfied, all future stages are **Revealed**
  - If sessions have non-forward-secret stages, those stages are **Revealed**

Security Definitions

- Security Definitions split into two separate games
  - **Match Security**
  - **Multi-Stage** (Key-Indistinguishability)

- Lots of other security properties to consider!
  - Forward secrecy
  - Increasing authentication levels
  - Key independence/dependence

Examples


ES
SS
0
Ext
0
Ext
H_1
tk_{hs}
Exp
xES
xSS
H_3
Exp
H_3
Exp
mES
mSS
Ext
Exp
H_3
H_{sess}
tk_{app}
Exp
MS
FS
H_{sess}
H_{sess}
EMS
Exp
Exp
RMS
(resumption)

Match Security Definitions

- **Match Security** captures the correctness of honest sessions
  - Also used for composability argument!
- Session identifiers are defined per-protocol basis
- **Match Security** (in each stage):
  - $\pi.sid = \pi'.sid \rightarrow \pi.key = \pi'.key$
  - $\pi.sid = \pi'.sid \rightarrow \pi.auth = \pi'.auth$
  - $\pi.sid = \pi'.sid \rightarrow \pi.cid = \pi'.cid$
  - $\pi.sid = \pi'.sid \rightarrow \pi.U = \pi'.V$ and $\pi.V = \pi'.U$ (depending on auth levels)
  - $\pi.sid_i \neq \pi.sid_j \; \forall \, i \neq j$
  - $\pi.sid = \pi'.sid \rightarrow \pi.sid = \pi'.sid \neq \pi''.sid$

Key Indistinguishability Definitions

- **Multi-Stage** Key-Indistinguishability game
  - Proof technique: Parse into three separate cases
  - Client accepts without matching Server in the first stage
    - Have to focus on server-authenticated keys due to non-matching first stage
    - Can reduce to assumptions $\text{Hash}_{\text{Coll}}$ and $\text{Sig}_{\text{EUF-CMA}}$
  - Adversary breaks game where Server accepts without matching Client
    - Assume mutual authentication (because sessions don't match!)
    - Can reduce to assumptions $\text{Hash}_{\text{Coll}}$ and $\text{Sig}_{\text{EUF-CMA}}$
  - Adversary breaks game when session accepts with matching session
    - Prove key-indistinguishability notions via standard assumptions

Examples

Client $\mathcal{C}$		Server $\mathcal{S}$
server's static public key $pk_{\mathcal{S}}$, $[\mathsf{nonce}_{\mathcal{S}}]$		server's static secret key $sk_{\mathcal{S}}$
generate ephemeral keys $\mathsf{esk}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}$		
generate $\mathsf{nonce}_{\mathcal{C}}$	$\xrightarrow{\mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{aux}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}}$	
$D_1 = \mathrm{DH}(\mathsf{esk}_{\mathcal{C}}, pk_{\mathcal{S}})$		$D_1 = \mathrm{DH}(\mathsf{epk}_{\mathcal{C}}, sk_{\mathcal{S}})$
$\mathsf{PRK}_1 = \mathrm{KDF}_{\mathrm{ext}}(D_1, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$		$\mathsf{PRK}_1 = \mathrm{KDF}_{\mathrm{ext}}(D_1, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$
$\mathsf{K}_1 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_1, \mathsf{info}_1)$		$\mathsf{K}_1 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_1, \mathsf{info}_1)$
	$\xleftarrow{\{\mathsf{aux}_{\mathcal{S}}, \mathsf{tpk}_{\mathcal{S}}\}_{\mathsf{K}_1}}$	use temporary keys $\mathsf{tsk}_{\mathcal{S}}, \mathsf{tpk}_{\mathcal{S}}$
$D_2 = \mathrm{DH}(\mathsf{esk}_{\mathcal{C}}, \mathsf{tpk}_{\mathcal{S}})$		$D_2 = \mathrm{DH}(\mathsf{epk}_{\mathcal{C}}, \mathsf{tsk}_{\mathcal{S}})$
$\mathsf{PRK}_2 = \mathrm{KDF}_{\mathrm{ext}}(D_2, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$		$\mathsf{PRK}_2 = \mathrm{KDF}_{\mathrm{ext}}(D_2, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$
$\mathsf{K}_2 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_2, \mathsf{info}_2)$		$\mathsf{K}_2 = \mathrm{KDF}_{\mathrm{exp}}(\mathsf{PRK}_2, \mathsf{info}_2)$

Figure 3: Expanded description of protocol run of Google's QUIC with 0-RTT handshake.

Client		Server
ClientHello: $r_c \leftarrow_{\$} \{0,1\}^{128}$ + ClientKeyShare: $X \leftarrow g^x$	$\longrightarrow$	
		ServerHello: $r_s \leftarrow_{\$} \{0,1\}^{128}$ + ServerKeyShare: $Y \leftarrow g^y$
	$\longleftarrow$	
	$H_1 \leftarrow \mathsf{H}(\mathtt{CH}\|\mathtt{SH})$ (incl. CKS & SKS)	
$\mathrm{ES} \leftarrow Y^x$		$\mathrm{ES} \leftarrow X^y$
	$\mathrm{xES} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{ES})$	
	$tk_{hs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_1 \| H_1)$ stage 1	
		{EncryptedExtensions}
		{ServerConfiguration*}
		{ServerCertificate*}: pk_S
		{CertificateRequest*}
	$H_2 \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CR}^*)$	
		{ServerCertificateVerify*}:
		$\mathtt{SCV} \leftarrow \mathsf{Sign}(sk_S, H_2)$
$\mathrm{SS} \leftarrow Y^x$		$\mathrm{SS} \leftarrow X^y$
	$\mathrm{xSS} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{SS})$	
	$H_3 \leftarrow \mathsf{H}(\mathtt{CH}\| \cdots \|\mathtt{SCV}^*)$	
	$\mathrm{FS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_2 \| H_3)$	
		{ServerFinished}:
		$\mathtt{SF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3 \| H_3)$
	$\longleftarrow$	
check $\mathsf{Verify}(pk_S, H_2, \mathtt{SCV}) = 1$		
check $\mathtt{SF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3 \| H_3)$		
{ClientCertificate*}: pk_C		
	$H_4 \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CCRT}^*)$	
{ClientCertificateVerify*}:		
$\mathtt{CCV} \leftarrow \mathsf{Sign}(sk_C, H_4)$		
	$H_{\mathrm{sess}} \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CCV}^*)$	
{ClientFinished}:		
$\mathtt{CF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4 \| H_{\mathrm{sess}})$	$\longrightarrow$	
		check $\mathsf{Verify}(pk_C, H_4, \mathtt{CCV}) = 1$
		check $\mathtt{CF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4 \| H_{\mathrm{sess}})$
	$\mathrm{mES} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_5 \| H_3)$	
	$\mathrm{mSS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_6 \| H_3)$	
	$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{mSS}, \mathrm{mES})$	
	$tk_{app} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_7 \| H_{\mathrm{sess}})$ stage 2	
		[NewSessionTicket$^{\Delta}$]: psk_id
	$\mathrm{RMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_8 \| H_{\mathrm{sess}})$ stage 3	
	$\mathrm{EMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_9 \| H_{\mathrm{sess}})$ stage 4	
record layer (application data), using AEAD with key tk_{app}		
$\longleftrightarrow$		

Reducing Multiple Test queries…

- Hang on, we only looked at a single **Test** case! But the model says we can **Test** lots of sessions!

- Correct! We use a hybrid argument to restrict an adversary from multiple **Test** queries to a single **Test** query
  - This reduces the adversary probability of success by a factor of $1/(P * n_s)$
  - P = number of stages; n_s = number of sessions
  - How does it work?

… to single Test query

- Imagine $P * n_s$ games, (allowing multiple **Test** queries) where in each game we increase the number of honestly generated keys as responses to **Test** queries
- The first game is equivalent to test bit = 1, last game to test bit = 0
- Simulate multi-**Test** game for adversary, pick a random index $i \in [0, P * n_s]$
  - For first $i - 1$ **Test** queries, return real keys – issue **Reveal** query to single-**Test** game
  - For i -th **Test** query, actually issue **Test** query to the single-**Test** game
  - For the rest of the **Test** queries, return random keys
- Can find the claim by a standard counting argument

Composability

- We can show composability of (multi-stage) key-exchange protocols and symmetric-key protocols
  - (I say we, it heavily follows BR key-exchange composition proved in [BFWW11])
- What does composability actually mean?
  - Fix a key exchange protocol **KE** with an arbitrary symmetric protocol Π
  - Adversary is to break security of Π using oracle queries to both **KE** and Π
  - Whenever session keys are derived in **KE**, they can be used in Π
  - BFWW11 show that session-matching necessary and sufficient condition for composability in BR-styled protocols

Composability II

- How do unauthenticated keys get treated?
  - As before, if not honestly contributed, we cannot expect security from Π
- Multi-stage session matching problem: First-stage keys encrypt parts of handshake, but we require public-session matching
- Solution: independently treat stages and utilize key-independence property:
  - **Reveal** previous stage key to decrypt handshake and get unencrypted messages

Composition Theorem

- Multi-Stage Composition Theorem:
  - If **KE** is stage-j forward secret, has key independence, multi-stage session matching and has stage-i keys established <u>after</u> the final messages are sent
    - (PS $i \leq j$)
  - Then the security of the composed game is bound by the:
    - Match security of **KE**
    - Multi-Stage security of **KE**
    - Security of $\boldsymbol{\Pi}$

Composability Proof

- Proof technique very similar to hybrid argument
  - Establish n_s games, where each game incrementally replaces keys derived by **KE** used in $\boldsymbol{\Pi}$ with random keys
  - First game all keys are real, last game all keys are random
  - We can bound the difference between the subsequent games by Prob(Multi-Stage)
  - Counting argument then gives us the bound between the first and last game by $n_s *$ Prob(Multi-Stage)

Authenticated and Confidential Channel Establishment Protocols

Overview

- Introduced in [JKSS12]
- Addresses “TLS Finished Message” problem
  - How can you create an RoR game where the key is utilized in the protocol?

- SOLUTION: Don’t use a RoR game!
  - Compose a stateful AE algorithm
  - Play ciphersuite indistinguishability game instead!
  - (Was this the right approach?)

Overview II

- Consists of 2 phases: Pre-Accept and Post-Accept
  - Pre-Accept = handshake
  - Post-Accept = AE channel
- Adversarial Interaction
  - **Send**: Send query for pre-accept sessions
  - **Reveal**: Standard query for leaking derived session keys
  - **Corrupt**: Standard query for leaking long-term keys
  - **Encrypt**: Plays the ciphertext indistinguishability game using session keys from HS
  - **Decrypt**: Send query for post-accept sessions, consistent with Encrypt

Security Definitions

- Security Definition is broken up into two sub-conditions
  - Authentication: Adversary can force a session to accept maliciously
    - i.e. without a matching conversation partner
    - No **Corrupt** before acceptance
    - No **Reveal** to matching partner session (if it exists)
  - Indistinguishability: Adversary can guess the test bit of a session
    - No **Corrupt** to partner before acceptance
    - No **Reveal** to matching partner session (if it exists)

Multi-Ciphersuite ACCE

- Why feel the need to complicate things?
- ACCE analysis assumes long-term keys in isolation
  - However, real life tells us otherwise
  - Mavrogiannopoulos et.al. [MVVP12] attack on TLS uses shared signing keys across ciphersuites in order to force a session to misinterpret ECDHE key shares as DHE

Multi-Ciphersuite ACCE II

- Idea: Break up the handshake + AE scheme into two separate phases
  - Negotiation Protocol (NP)
    - Common in all uses of the protocol, used to negotiate a ciphersuite
    - Has algorithms $NP.ALG_{init}$ and $NP.ALG_{resp}$
  - Sub-Protocols (**SP**)
    - The ciphersuites to be chosen from
    - Has algorithms SP.KeyGen, SP.Enc, SP.Dec and $SP.ALG_{init}$ and $SP.ALG_{resp}$
  - A Multi-Ciphersuite Protocol is the collection of NP||**SP**

Multi-Ciphersuite ACCE III

- Intention: To create a composition theorem
  - Prove ACCE security with additional oracles
  - Then prove sub-conditions
- We should now have security even when reusing long-term keys
- Intuition: If you sign anything, you should include the identifier of the ciphersuite
- Implication: Can't mix up signatures between sub-protocols
  - It follows that if a ciphersuite is broken, sharing keys did not help the adversary

Multi-Ciphersuite ACCE IV

- Security experiment: Run an $ACCE_Z$ challenger with auxiliary oracle
  - Aux(*sk*,*x*) -> *y* performs signature on *x* using *sk* if predicate doesn't hold
  - Predicate = contains identifier of the ciphersuite Z
- If the adversary forces negotiation to ciphersuite Z
  - Forward all queries to $ACCE_Z$ challenger
- Need to be able to simulate other ciphersuites
  - If key isn't shared for that given ciphersuite, the simulator generates it's own keys
  - If key is shared, use Aux oracle to obtain necessary signatures
- If you break Multi-Ciphersuite Security, you also break ACCE-Aux security of the underlying ciphersuite

Negotiable ACCE

- Modern protocol implementations have complex functionality
- Algorithmic agility is desired to increase interoperability
- Interoperability can affect security
- Version downgrade attacks possible on SSLv2
- Legacy crypto opens potential for further downgrade attacks

Negotiation

- Much like for Multi-Ciphersuite ACCE, treat handshake as two separate phases
- Negotiation Phase and Sub-Protocol Phase
- Describe an optimality function for negotiation
  - Both parties have an ordered list of ciphersuite preferences
  - Both have access to optimality function
- Optimal Negotiation
  - Both parties output the same value
  - The negotiated value is the output of the optimality function

Negotiation-Authentication Theorem

- Intuitively, downgrade attacks break authentication and force sub-optimal negotiation, thus we should be able to relate the two

- Negotiation-Authentication Theorem:
  - Condition One: All Negotiation Phase messages are in a session identifier
  - Condition Two: In the absence of an active adversary, negotiation is optimal

$$\mathrm{Adv}^{\mathsf{neg},\omega}_{\mathtt{NP}\|\overrightarrow{\mathtt{SP}},n}(\mathcal{A}) = \mathrm{Adv}^{\mathsf{acce\text{-}auth}}_{\mathtt{NP}\|\mathtt{SP}_n}(\mathcal{A})$$

Considerations to miTLS

- Relating security results for miTLS project to other security models
- Equivalence of a key-idealization model to a Test query model?
- “Lifting” security properties
  - miTLS considers Passive Key Indistinguishability and Authentication
  - Is this equivalent to MSKE’s Multi-Stage security definitions?
  - Can we prove statements about freshness predicates?
- Definitions of session/contributive identifiers in MSKE are very coarse
  - Can we show more fine-grained security properties?
  - Imagine cids as purely keyshare information
  - Defining sids as keys?

Thank you!

Any questions?