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.

Modelling ciphersuite and version negotiation in the TLS protocol.

Multi-ciphersuite security of the Secure Shell (SSH) protocol.
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 $C$
server’s static public key $pk_S$, $\{\text{nonce}_S\}$
generate ephemeral keys $esk_C, epk_C$
generate $\text{nonce}_C$

\[ D_1 = \text{DH}(esk_C, pk_S) \]

\[ \text{PRK}_1 = \text{KDF}_{\text{ext}}(D_1, \text{nonce}_C, \text{nonce}_S) \]

\[ K_1 = \text{KDF}_{\text{exp}}(\text{PRK}_1, \text{info}_1) \]

\[ D_2 = \text{DH}(esk_C, \text{tpk}_S) \]

\[ \text{PRK}_2 = \text{KDF}_{\text{ext}}(D_2, \text{nonce}_C, \text{nonce}_S) \]

\[ K_2 = \text{KDF}_{\text{exp}}(\text{PRK}_2, \text{info}_2) \]

\[ [\text{aux}_S, \text{tpk}_S] = K_1 \]

\[ \text{Server } S \]
server’s static secret key $sk_S$

\[ D_1 = \text{DH}(epk_C, sk_S) \]

\[ \text{PRK}_1 = \text{KDF}_{\text{ext}}(D_1, \text{nonce}_C, \text{nonce}_S) \]

\[ K_1 = \text{KDF}_{\text{exp}}(\text{PRK}_1, \text{info}_1) \]

\[ \text{use temporary keys } tsk_s, \text{tpk}_s \]

\[ D_2 = \text{DH}(epk_C, tsk_s) \]

\[ \text{PRK}_2 = \text{KDF}_{\text{ext}}(D_2, \text{nonce}_C, \text{nonce}_S) \]

\[ K_2 = \text{KDF}_{\text{exp}}(\text{PRK}_2, \text{info}_2) \]

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

Client $C$
server’s static public key $pk_S$, [nonces$_C$]
generate ephemeral keys $esk_C, epk_C$
generate nonce$_C$
$D_1 = DH esk_C, pk_S$
$PRK_1 = \text{KDF}_\text{ext}(D_1, \text{nonce}_C, [\text{nonces}_C])$
$K_1 = \text{KDF}_\text{exp}(PRK_1, \text{info}_1)$
$D_2 = DH esk_C, tpk_S$
$PRK_2 = \text{KDF}_\text{ext}(D_2, \text{nonce}_C, [\text{nonces}_C])$
$K_2 = \text{KDF}_\text{exp}(PRK_2, \text{info}_2)$

$\text{nonces}_C, [\text{nonces}_C], aux_c, epk_c, K_1, K_2, \text{aux}_S, \text{tpk}_S$

Server $S$
sender’s static secret key $sk_S$
$D_1 = DH(epk_C, sk_S)$
$PRK_1 = \text{KDF}_\text{ext}(D_1, \text{nonce}_C, [\text{nonces}_C])$
$K_1 = \text{KDF}_\text{exp}(PRK_1, \text{info}_1)$
use temporary keys $tks_c, tpk_S$
$D_2 = DH(epk_C, tsk_S)$
$PRK_2 = \text{KDF}_\text{ext}(D_2, \text{nonce}_C, [\text{nonces}_C])$
$K_2 = \text{KDF}_\text{exp}(PRK_2, \text{info}_2)$

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

ClientHello: $r_s \leftarrow \{0,1\}^{128}$
+ ClientKeyShare: $X \leftarrow g^x$

ServerHello: $r_s \leftarrow \{0,1\}^{128}$
+ ServerKeyShare: $Y \leftarrow g^y$

$H_1 \leftarrow H(\text{CH}||\text{SH})$ (incl. CKS & SKS)
$ES \leftarrow Y^x$

t$_{sh} \leftarrow \text{HKDF}_\text{Expand}((0, ES), t_{sh}, H_{\text{HMAC}})$

EncryptExtensions
+ ServerConfiguration
+ ServerCertificate
+ CertificateRequest

$H_2 \leftarrow H(\text{CH}||\text{SH}||\text{SS}||\text{SCV})$

$SS \leftarrow Y^y$

t$_{ss} \leftarrow \text{HKDF}_\text{Expand}((0, SS), t_{ss}, H_{\text{HMAC}})$

FS \leftarrow \text{HKDF}_\text{Expand}(t_{ss}, H_{\text{HMAC}})$

ServerFinished:

Check Verify($pk_S, H_2, SCV) = 1$
Check $SF = \text{HMAC} (FS, $label$_s||H_3)$

{ClientCertificate}:
$pkc$
$H_4 \leftarrow H(\text{CH}||\text{CCRT})$

{ClientCertificateVerify}:
$CCV \leftarrow \text{Sign}(sk_S, H_4)$
$H_{\text{seen}} \leftarrow H(\text{CH}||\text{CCRT})$

{ClientFinished}:
$CF \leftarrow \text{HMAC} (FS, $label$_s||H_{\text{seen}})$

Check Verify($pk_C, H_4, CCV) = 1$
Check $CF = \text{HMAC} (FS, $label$_s||H_{\text{seen}})$

$mES \leftarrow \text{HKDF}_\text{Expand}(mES, $label$_s||H_3)$

$mSS \leftarrow \text{HKDF}_\text{Expand}(mES, $label$_s||H_3)$

$MS \leftarrow \text{HKDF}_\text{Expand}(mSS, mES)$

$t_{app} \leftarrow \text{HKDF}_\text{Expand} (MS, $label$_s||H_{\text{seen}})$

record layer (application data), using AEAD with key $t_{app}$
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 $C$
server’s static public key $pk_S$, $\langle$nonce$\rangle_S$
generate ephemeral keys $esk_c, epk_c$
generate nonce$\langle$nonce$\rangle_c$

$$D_1 = DH(esk_c, pk_S)$$

$$PRK_1 = KDF_{ext}(D_1, \text{nonce}_c, \text{nonce}_S)$$

$$K_1 = KDF_{exp}(PRK_1, \text{info}_1)$$

$$D_2 = DH(esk_c, \text{tpk}_S)$$

$$PRK_2 = KDF_{ext}(D_2, \text{nonce}_c, \text{nonce}_S)$$

$$K_2 = KDF_{exp}(PRK_2, \text{info}_2)$$

$[\text{aux}_c, \text{tpk}_c]_k_1$

Server $S$
server’s static secret key $sk_S$

$$D_1 = DH(epk_c, sk_S)$$

$$PRK_1 = KDF_{ext}(D_1, \text{nonce}_c, \text{nonce}_S)$$

$$K_1 = KDF_{exp}(PRK_1, \text{info}_1)$$

use temporary keys $\text{tsk}_S, \text{tpk}_S$

$$D_2 = DH(epk_c, \text{tsk})$$

$$PRK_2 = KDF_{ext}(D_2, \text{nonce}_c, \text{nonce}_S)$$

$$K_2 = KDF_{exp}(PRK_2, \text{info}_2)$$

Figure 3: Expanded description of protocol run of Google’s QUIC with 0-RTT handshake.
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 C**
server’s static public key $pk_S$, [nonces]
generate ephemeral keys $esk_C, epk_C$
generate $\text{nonces}_C$
$D_1 = \text{DH}(esk_C, pk_S)$
$\text{PRK}_1 = \text{KDF}_\text{ext}(D_1, \text{nonces}_C, [\text{nonces}_S])$
$K_1 = \text{KDF}_\text{exp}(\text{PRK}_1, \text{info}_1)$
$D_2 = \text{DH}(esk_C, tpk_S)$
$\text{PRK}_2 = \text{KDF}_\text{ext}(D_2, \text{nonces}_C, [\text{nonces}_S])$
$K_2 = \text{KDF}_\text{exp}(\text{PRK}_2, \text{info}_2)$

**Server S**
server’s static secret key $sk_S$
$D_1 = \text{DH}(epk_C, sk_S)$
$\text{PRK}_1 = \text{KDF}_\text{ext}(D_1, \text{nonces}_C, [\text{nonces}_S])$
$K_1 = \text{KDF}_\text{exp}(\text{PRK}_1, \text{info}_1)$
use temporary keys $tks_S, tpk_S$
$D_2 = \text{DH}(epk_C, tks_S)$
$\text{PRK}_2 = \text{KDF}_\text{ext}(D_2, \text{nonces}_C, [\text{nonces}_S])$
$K_2 = \text{KDF}_\text{exp}(\text{PRK}_2, \text{info}_2)$

---

Figure 3: Expanded description of protocol run of Google’s QUIC with 0-RTT handshake.
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 cannot compromise semi-ephemeral secrets
  - Security of TLS handshake (except 0-RTT) independent of these secrets

- Adversary cannot 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
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):
  - $\pi.\text{sid} = \pi'.\text{sid} \rightarrow \pi.\text{key} = \pi'.\text{key}$
  - $\pi.\text{sid} = \pi'.\text{sid} \rightarrow \pi.\text{auth} = \pi'.\text{auth}$
  - $\pi.\text{sid} = \pi'.\text{sid} \rightarrow \pi.\text{cid} = \pi'.\text{cid}$
  - $\pi.\text{sid} = \pi'.\text{sid} \rightarrow \pi.\text{U} = \pi'.\text{V}$ and $\pi.\text{V} = \pi'.\text{U}$ (depending on auth levels)
  - $\pi.\text{sid}_i \neq \pi.\text{sid}_j \forall i \neq j$
  - $\pi.\text{sid} = \pi'.\text{sid} \rightarrow \pi.\text{sid} = \pi'.\text{sid} \neq \pi''.\text{sid}$
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** $C$
server’s static public key $pk_S$, [nonce$_S$]
generate ephemeral keys $esk_C$, $epk_C$
generate nonce$_C$
$D_1 = \text{DH}(esk_C, pk_S)$
$PRK_1 = \text{KDF}_{\text{ext}}(D_1, \text{nonce}_C, \text{nonce}_S)$
$K_1 = \text{KDF}_{\text{exp}}(PRK_1, \text{info}_1)$
$D_2 = \text{DH}(esk_C, tspk_S)$
$PRK_2 = \text{KDF}_{\text{ext}}(D_2, \text{nonce}_C, \text{nonce}_S)$
$K_2 = \text{KDF}_{\text{exp}}(PRK_2, \text{info}_2)$

**Server** $S$
server’s static secret key $sk_S$
Use temporary keys $tsk_C$, $tpsk_S$
$D_1 = \text{DH}(epk_C, sk_S)$
$PRK_1 = \text{KDF}_{\text{ext}}(D_1, \text{nonce}_C, \text{nonce}_S)$
$K_1 = \text{KDF}_{\text{exp}}(PRK_1, \text{info}_1)$
$D_2 = \text{DH}(epk_C, tsk_S)$
$PRK_2 = \text{KDF}_{\text{ext}}(D_2, \text{nonce}_C, \text{nonce}_S)$
$K_2 = \text{KDF}_{\text{exp}}(PRK_2, \text{info}_2)$

Figure 3: Expanded description of protocol run of Google’s QUIC with 0-RTT handshake.
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 $1/(P \times n_s)$
  - $P =$ number of stages; $n_s =$ number of sessions
  - How does it work?
... to single Test query

- Imagine $P \times n_s$ 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 $i \in [0, P \times n_s]$.
  - 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 \( \Pi \)
  - Adversary is to break security of \( \Pi \) using oracle queries to both KE and \( \Pi \)
  - Whenever session keys are derived in KE, they can be used in \( \Pi \)
  - 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 II

- 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 *after* 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 \( \Pi \)
Composability Proof

- Proof technique very similar to hybrid argument
  - Establish $n_s$ games, where each game incrementally replaces keys derived by KE used in $H$ 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 $\text{Prob}(\text{Multi-Stage})$
  - Counting argument then gives us the bound between the first and last game by $n_s \cdot \text{Prob}(\text{Multi-Stage})$
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 \textit{Corrupt} before acceptance
    - No \textit{Reveal} to matching partner session (if it exists)
  - Indistinguishability: Adversary can guess the test bit of a session
    - No \textit{Corrupt} to partner before acceptance
    - No \textit{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
  - \(\text{Aux}(sk,x) \rightarrow y\) 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**
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

\[
\text{Adv}^{\text{neg}, \omega}_{\text{NP} || \text{SP}, n}(A) = \text{Adv}^{\text{acce-auth}}_{\text{NP} || \text{SP}, n}(A)
\]
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?