OPTLS and TLS 1.3

Hugo Krawczyk, Hoeteck Wee

TRON Workshop
2/21/2016

Plan

- Explain OPTLS approach and modes (handshake only)
  - Highlight protocol concept and simplicity
  - Common logic to all protocol modes (helps analysis and maintenance)
  - Important feature: No new/fancy crypto, just careful engineering! (boring is good)
- Show how OPTLS modes translate into TLS 1.3 handshake modes
  - How the structure and approach (and analysis) of OPTLS still underlie TLS 1.3 and <u>why this is a good thing</u>.
- Mention the "key freshness" principle and why we should keep it
- Time permitting: Discuss KDF, Client authentication, SNI encryption

Motivating Requirements

- Forward secrecy, 0-RTT, ECC-centric (→ DH-based design)
- Simplicity, uniformity (minimize code flows, use KDF to drive modes), allow for performance optimizations
- Amenable to analysis: Uniform logic across different modes
  - DH and MAC-centric
- Easy to extend and maintain ("design robustness")

- Note: We only deal with the handshake protocol in this talk and ignore handshake encryption for now
  - It was "without loss of generality" till a few days ago and an annoying nuisance now (but not a game changer for this presentation)

OPTLS Starting Point (DH certs)

$$C \xrightarrow{\text{C-Hello},\ g^x} S$$

$$C \xleftarrow{\text{S-Hello},\ g^y,\ \text{S-Finished}} S$$

- S-Finished = $\mathrm{PRF}(g^{xs} \underbrace{;\ \text{transcript}}_{\text{nonces},\ g^y,\ \dots})$; g^{xs} defined via g^s (g^s to be defined)

OPTLS Starting Point (DH certs)

C $\xrightarrow{\text{C-Hello, } g^x\text{, [C-EarlyData]}}$ S

$\xleftarrow{\text{S-Hello, } g^y\text{, [DH-cert], S-Finished}}$

- S-Finished = PRF(g^{xs} ; transcript); g^{xs} defined via g^s (g^s to be defined)
- DH-cert: Server's identity, key g^s, CA signature on g^s and identity
- DH-cert can be omitted if client has cached key g^s
  - Caching enables 0-RTT: C-EarlyData = Enc(g^{xs} ; early-data)
- Omitted for now (as not essential for basic KE security):
  - DH-cert encryption and client's Finish (added later as important enhancers)

OPTLS with Online Signatures

C $\xrightarrow{\text{C-Hello, } g^x\text{, [C-EarlyData]}}$ S

C $\xleftarrow{\text{S-Hello, } g^y\text{, [}g^s\text{, sig] , S-Finished}}$ S

- DH-cert replaced by (g^s, sig) where sig = S-cert + Sig_S(g^s, nonces, …)
  - Nonces → Signature is fresh
- DH-cert logic applied here too but with fresh online signatures (instead of CA/offline ones)
  - Transcript authentication via S-Finish (sig → g^s → Finish → Transcript)

OPTLS with Ephemeral g^s

C $\xrightarrow{\text{C-Hello, } g^x\text{, [C-EarlyData]}}$ S

$\xleftarrow{\text{S-Hello, } g^y\text{, [} g^s \text{, sig] , S-Finished}}$

- DH-cert replaced by (g^s, sig) where sig = S-cert + $Sig_S(g^s$, nonces, …)
  - Observation: If g^s is ephemeral (used once) then protocol is still secure
  - Identifying g^s with g^y we get a mode without server's static key
    - g^y, $Sig_S(g^y$, nonces), S-Finished = PRF(g^{xy}; transcript) ("use-once static")
- Original DH-cert logic still applies ("uniform logic across modes")
  - Transcript authent'n via S-Finished (sig → $g^y \equiv g^s$ → Finish → Transcript)

Summary: OPTLS Modes

C → S: C-Hello, g^x, [C-EarlyData]

S → C: S-Hello, g^y, [g^s, sig], S-Finished

C-EarlyData: Enc(g^{xs} ; early-data)

[g^s, sig]: g^s, S-cert, $Sig_S(g^s, \text{nonces})$

S-Finished: PRF(g^{xs} ; transcript)

- Cached modes derive keys from both g^{xs} and g^{xy}, ephemeral only from g^{xy}
- Cached 1-RTT: Basic protocol only; Cached g^s; no early data (0 sig, 2 exp)
- Cached 0-RTT: Basic + C-EarlyData; Cached g^s; early data (0 sig, 2 exp)
- Ephemeral 1-RTT: Basic + [g^s, sig]; No caching; $g^s \leftarrow g^y$ (1 sig, 1 exp)
  - Optimal performance (TLS 1.3 "sacrifices" optimality with added signatures)
- Not in TLS 1.3: DH certs (DH-cert instead of [g^s, sig]) or its "offline sig" variant

OPTLS Extension for PSK Modes

- PSK = Pre-shared key mode, with and without PFS, and a basis for the session resumption mode:
  - Simply replace g^{xs} with PSK;   PSK → Finish → Transcript
  - The benefit of uniformity and Finished-based authentication

Uniformity: Server Authentication

- 0-RTT: cached g^s → Finish → Transcript
- 1-RTT: sig → g^s / g^y → Finish → Transcript
- PSK: PSK → Finish → Transcript
- (DH-cert: cert → g^s → Finish → Transcript)

OPTLS in TLS 1.3

- *Same* modes as OPTLS *augmented with:*
  - Signatures in all non-PSK modes (including cached modes)
    - Added for uniformity of specification and implementation
    - Not essential for basic KE security but adds value:
      - Shows continuous possession of signing key by server;
      - Helps against cross protocol attack [Jager et al] (RSA key dual use)
    - Costs extra signature in cached modes (cheap for ECDSA expensive for RSA)
  - Client Finished: Key confirmation (esp. to identify 0-RTT replay); UC security
  - KDF inputs: Minimalist(OPTLS), Maximalist in TLS 1.3 (robustness)
  - Finished key computed based on both g^{xs} and g^{xy} (requires tweak to analysis)

OPTLS in TLS 1.3 Handshake

- In spite of additions, the OPTLS underlying design is preserved
  - Particularly, the uniform logic (as well as the KDF)
- Important: OPTLS analysis still applicable to TLS 1.3
  - Even though TLS 1.3 now *looks* very signature oriented, OPTLS shows some of these signatures to be non-essential

"TLS 1.3 handshake = OPTLS in (signature) disguise"

Recent debate: Handshake traffic key = application traffic key ?

- Breaks key freshness/indistinguishability principle (not a *generic* KE)
- Important to keep modularity for design, analysis, maintanance
  - Would not change OPTLS applicability to TLS 1.3 but analysis needs to be adjusted (key exchange guarantee is *weakened*)

Beyond TLS 1.3

- OPTLS can inform future variants/changes/extensions/optimizations
- Potential TLS 1.3 extensions supported through OPTLS approach:
  - A *simple* DH-cert solution
  - With DH-based client auth'n, enables very efficient HMQV-like protocols
  - "Offline signature solution"
    - Server's DH cert replaced w/ signature cert plus (offline) signature on g^s
  - Post-quantum transition: Static QR encryption + ephemeral ECC DH
  - Cool SNI encryption solution

Concluding Remarks

- OPTLS unifying logic → design, analysis, extensions, maintenance
- Directly relevant to TLS 1.3 in spite of added signatures
- KDF at the service of streamlined code: Modes defined via key derivation (+HKDF: yet another unifying tool)
- Future: Will we see a *simple* DH-cert based solution implemented?
- Present: Will we go back to "key freshness"?

- Client authentication: Do we care about *deniability?*
  - Avoid signing the server's identity (requires care)
  - "SIGMAC Compiler"

Final Remark

- Ban proof-less crypto (though crypto with proofs is not failure-proof; need to be as *robust* as possible to misuse – the simpler the better)
- Bottom Up vs Top Down analysis
  - Bottom up (reductionist) approach: great "proof-driven" design tool and foundation for protocol logic; informs other tools; but "human-intensive" (prone to mistakes and can't handle high complexity) → OPTLS
  - Top down (automated) approach: Build on bottom up designs but can deal with more complexity and, most importantly, with the soundness of comprehensive specification and implementation → miTLS, Tamarin, …
  - Both approaches instrumental in ensuring a secure design
- OPTLS not intended as full design, or full analysis, of TLS 1.3 but to inform its core crypto design (much left out; e.g. mode composition)

Thanks!

OPTLS: http://eprint.iacr.org/2015/978

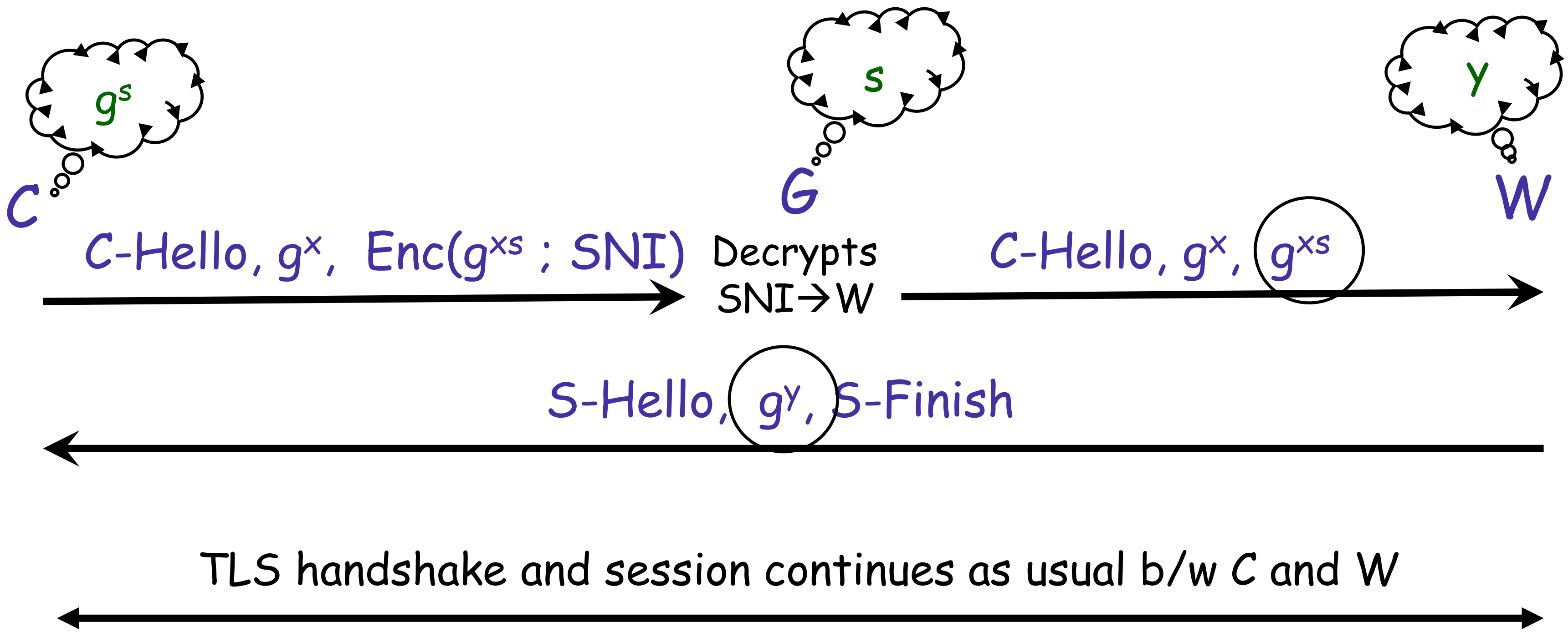
Notes on KDF

- KDF: Not covered here (would need another ½ hour)
- But a fundamental piece in OPTLS and TLS 1.3 design (driver for different modes – a uniform derivation path, via value setting)
- The ultimate example of HKDF design rationale:
  - It uses the full range of functionalities: Extraction, Expand, PRF, RO
  - All under the same primitive and flexible for different analyses (e.g. RO)
- Example: master_secret = KDF(salt=g^{xs}, source=g^{xy})
  - If g^{xs} secure then HMAC as PRF, if g^{xs} leaked then HMAC as Extractor
  - Compare with master_secret = H(g^{xs}) xor H(g^{xy}) when g^{xs}=g^{xy}

SIGMAC: Privacy-Friendly Client Authentication

- A *compiler* from unilateral-to-mutual authentication
- Applicable to client authentication in TLS 1.3 (including post-handshake)
- Avoids signing the server's identity (by the client)
- Raises some unexpected subtleties (need for including S-Finished under client's signature is one of them)
- Follows the SIGMA ("SIGn-and-Mac") approach
- SIGMAC: Add the following to a server-authenticated KE:
  - Signature: Client signs parts of the transcript (complier tells you what), without including the server identity
  - MAC: Include under client's Finished the client's and server's identities

SNI Encryption using OPTLS



- C can compute key material since it knows x, g^s, g^y;
- W can compute it since it knows g^x, y, g^{xs}
- G cannot read traffic as it does not have y