A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello

Karthikeyan Bhargavan, Vincent Cheval, Christopher Wood*

Inria Paris, Cloudflare*

CCS '22

2022.11.21.

GyeongHeon Jeong(ghjeong@mmlab.snu.ac.kr)

Index

- Introduction
- Background
  - Basic TLS 1.3
  - TLS 1.3 with All Features
- Security Goals
- Encrypted ClientHello (ECH)
- Model & Result
- Conclusion

Introduction

- **TLS 1.3**, the newest version of the Transport Layer Security (TLS) protocol, yet **privacy** guarantees of TLS 1.3 remain weak and poorly understood
    - The protocol reveals the identity of the target server allowing the passive surveillance and active censorship of TLS connections

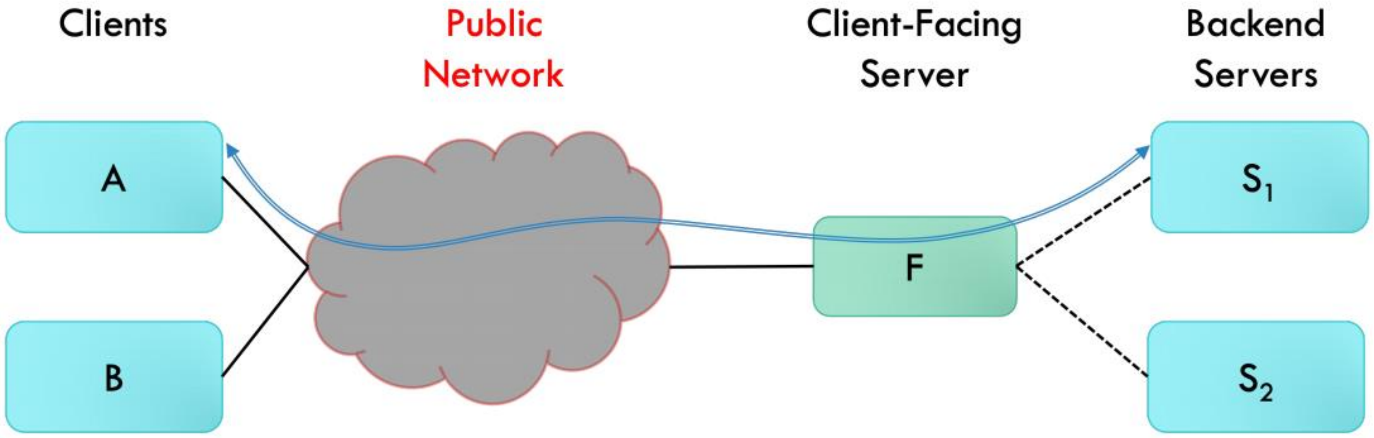
- To close this gap, the IETF TLS working group is standardizing a new privacy extension called **Encrypted Client Hello** (**ECH**)
    - The absence of a formal privacy model makes it hard to verify that this extension works

- This paper presents the first mechanized formal analysis of privacy properties for the TLS 1.3 handshake
    - Using the symbolic protocol analyzer **ProVerif**
    - One of the largest privacy proofs attempted using an automated verification tool

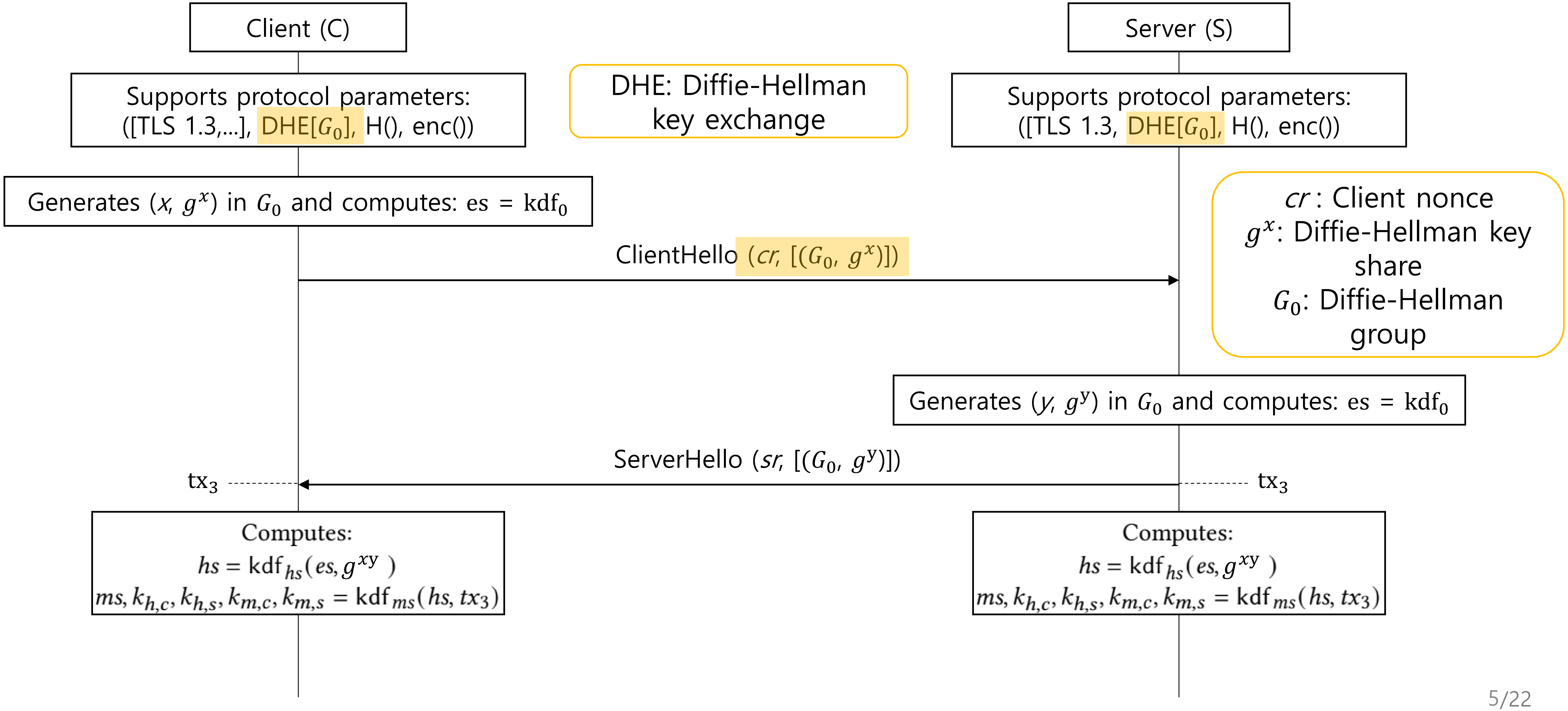
Background

- This paper shows all standard modes of TLS 1.3, with and without ECH
  - Therefore, the detail explanation of TLS 1.3 and ECH will be followed

- Typical TLS 1.3 deployment scenario
  - Two clients: A, B (e.g., Web browsers)
  - Backend servers: S_1, S_2 (e.g., Websites)
  - Client-facing server: F (e.g., Content delivery network)



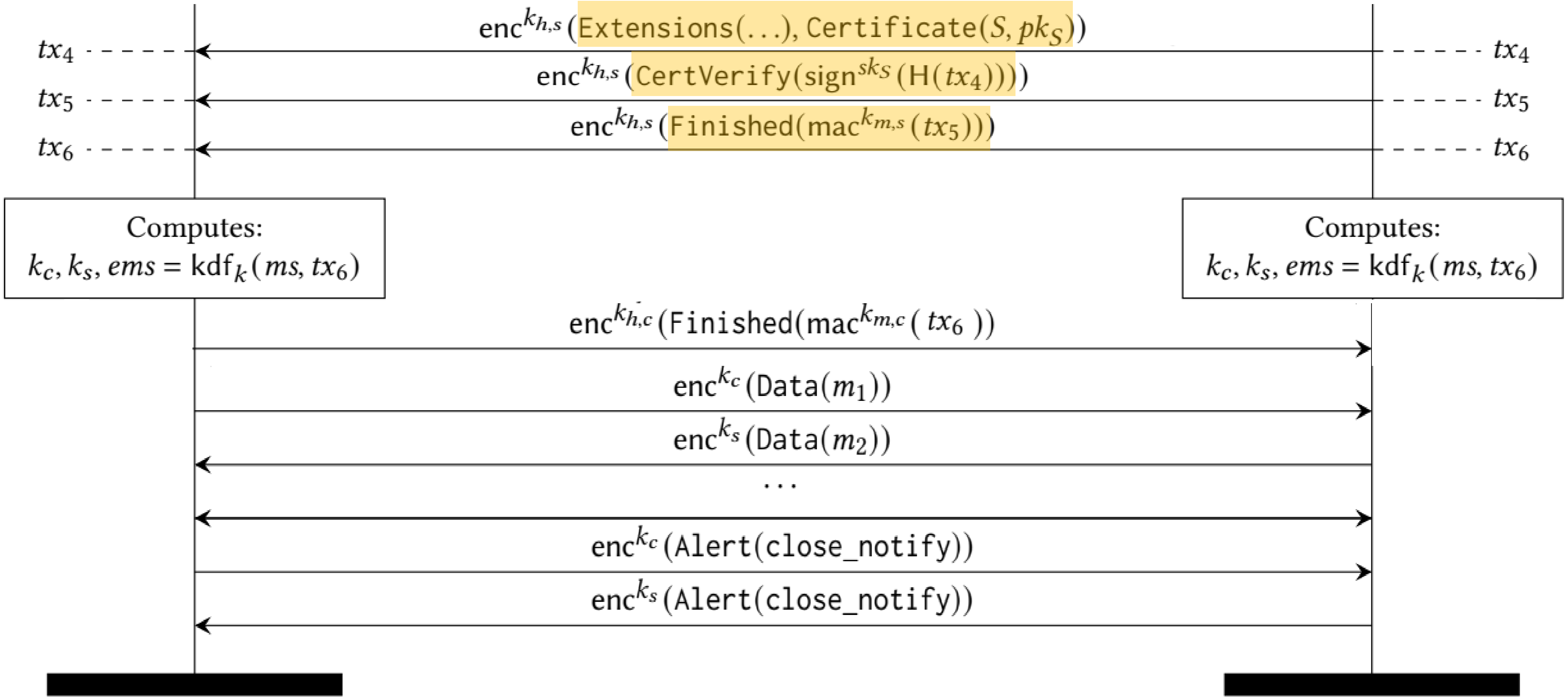
Basic TLS 1.3



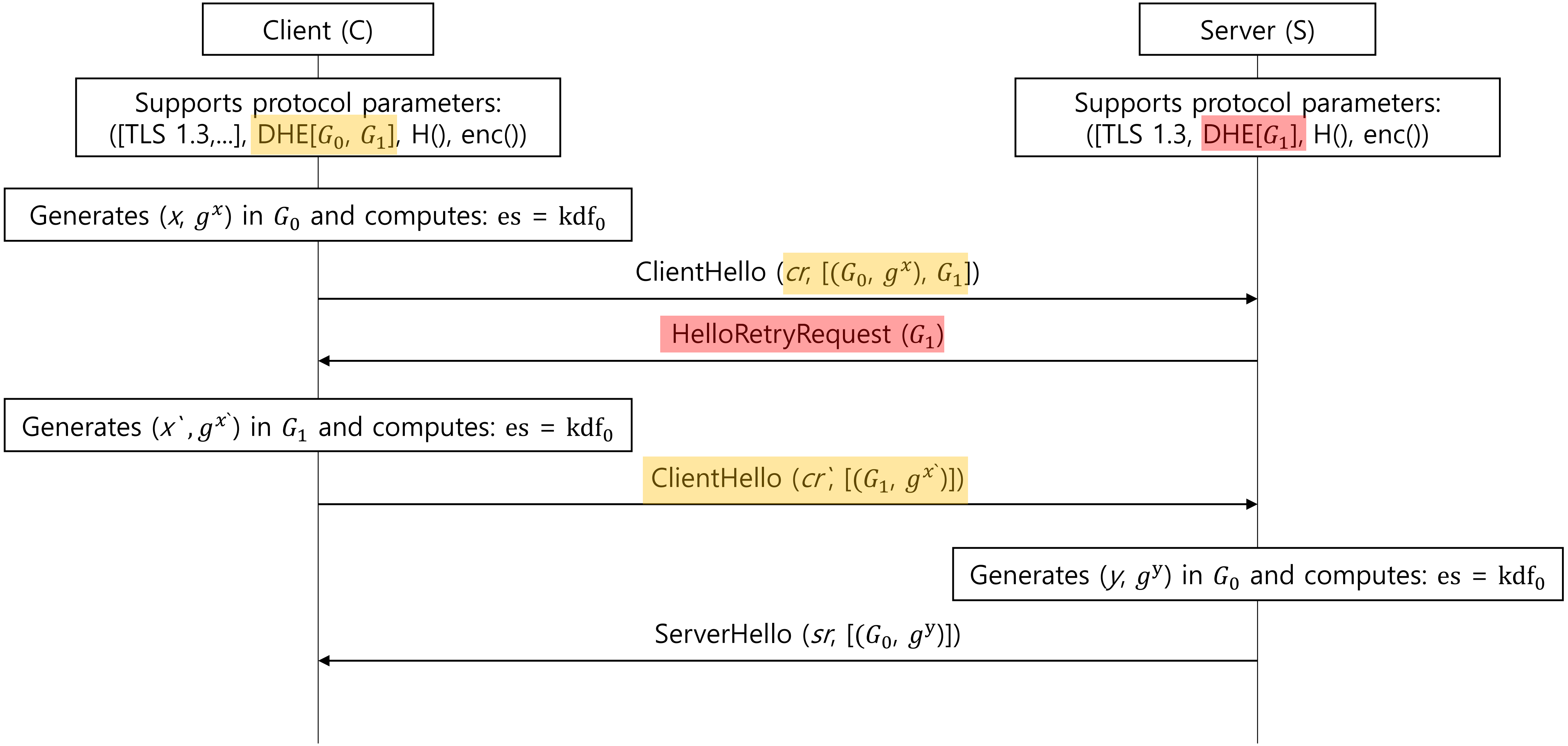
Client (C)
Server (S)
Supports protocol parameters: ([TLS 1.3,...], DHE[G_0], H(), enc())
DHE: Diffie-Hellman key exchange
Supports protocol parameters: ([TLS 1.3, DHE[G_0], H(), enc())
Generates (x, g^x) in G_0 and computes: es = kdf_0
cr : Client nonce
g^x: Diffie-Hellman key share
G_0: Diffie-Hellman group
ClientHello (cr, [(G_0, g^x)])
Generates (y, g^y) in G_0 and computes: es = kdf_0
ServerHello (sr, [(G_0, g^y)])
tx_3
tx_3
Computes:
$hs = \mathrm{kdf}_{hs}(es, g^{xy})$
$ms, k_{h,c}, k_{h,s}, k_{m,c}, k_{m,s} = \mathrm{kdf}_{ms}(hs, tx_3)$
Computes:
$hs = \mathrm{kdf}_{hs}(es, g^{xy})$
$ms, k_{h,c}, k_{h,s}, k_{m,c}, k_{m,s} = \mathrm{kdf}_{ms}(hs, tx_3)$

Basic TLS 1.3

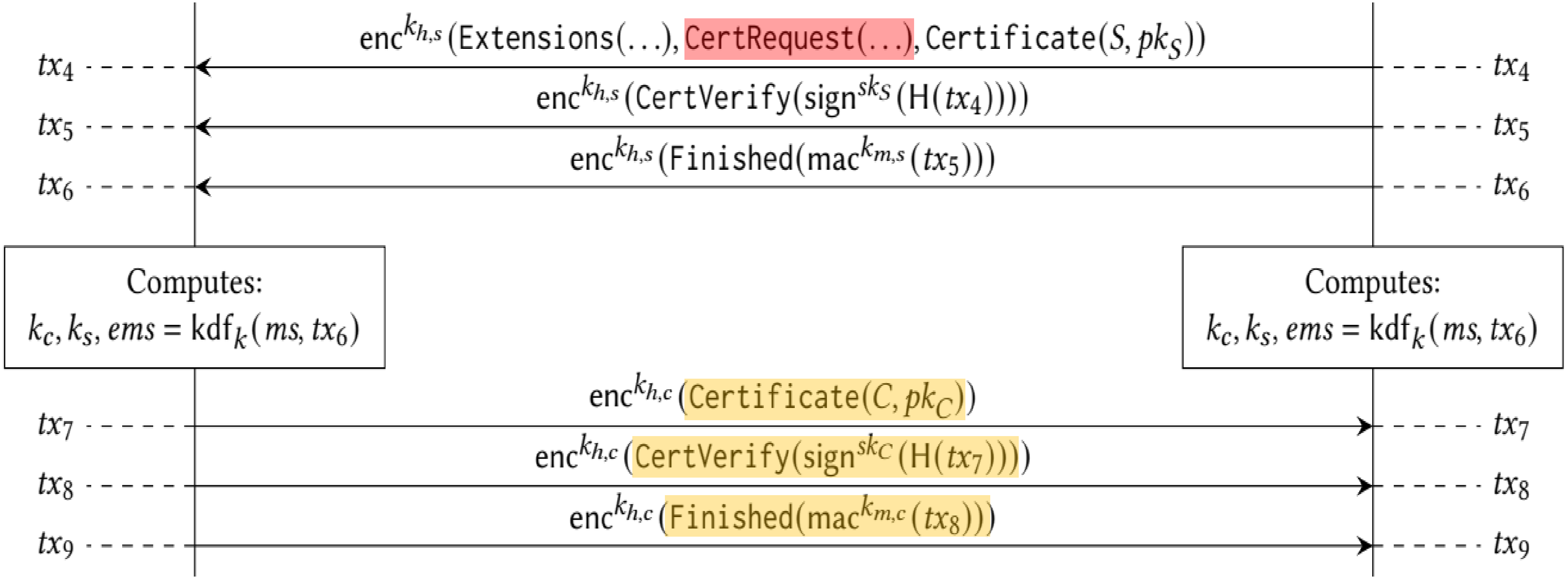
Extensions: contains additional server parameters
Certificate: contains the server's public-key certificate
CertVerify: contains a signature over the handshake transcript so far over server's private-key
Finished: contains a MAC over the handshake transcript up to CertVerify



Negotiating Connection Parameters



Certificate-based Client Authentication



Pre-Shared Keys (PSK)

- If the client and server have been configured with a pre-shared symmetric key ($psk_{C,S}$), then they can instead use this PSK to authenticate each other
  - External PSK provided by the application
  - Resumption PSK output by a prior handshake between the client and server
- After the end of each handshake, the server may send the client a session ticket (**SessionTicket**) that serves as a new PSK identifier (*psk`*)
  - Save it for use in next PSK handshakes
- In a PSK handshake, the client already has a key it shares with the server, and so it can start sending data immediately after the ClientHello message without waiting
  - This data is called **0-RTT Data**

TLS Extensions

- ClientHello message indicate protocol extensions that the client supports, and the server may choose some of these extensions in the ServerHello

- **Server Name Indication (SNI)**:
  - Most common TLS extension on the Web
  - The ClientHello includes the name of the server to which the client wishes to connect
  - Needed by web hosts and content-delivery networks that host multiple domains and have to decide which server to use for each connection

- By default, all extensions sent in the ClientHello, ServerHello messages are **unencrypted**, but the server can encrypt some extension data in its Extensions message
  - ECH extension allows the client to also encrypt elements of the ClientHello, including the SNI extension

Security Goals of TLS 1.3

Authentication and Integrity Goals	Verification Tool
Server Authentication (SAUTH)	(1,3,4)
Client Authentication (CAUTH)	(1,3,4)
Key and Transcript Agreement (AGR)	(1,3,4)
Data Stream Integrity (INT)	(1,2,3,4)
Key Uniqueness (UNIQ)	(3,4)
Downgrade Resilience (DOWN)	(4)
Confidentiality	
Key Secrecy (SEC)	(1,2,3,4)
Key Indistinguishability (IND)	(1)
1-RTT Data Forward Secrecy (FS)	(1,3,4)
0-RTT Data Secrecy (SEC0)	(1,2,3,4)

1. CryptoVerif
2. F*
3. Tamarin
4. ProVerif

These models do not cover all features

Security Goals of TLS 1.3 (cont.)

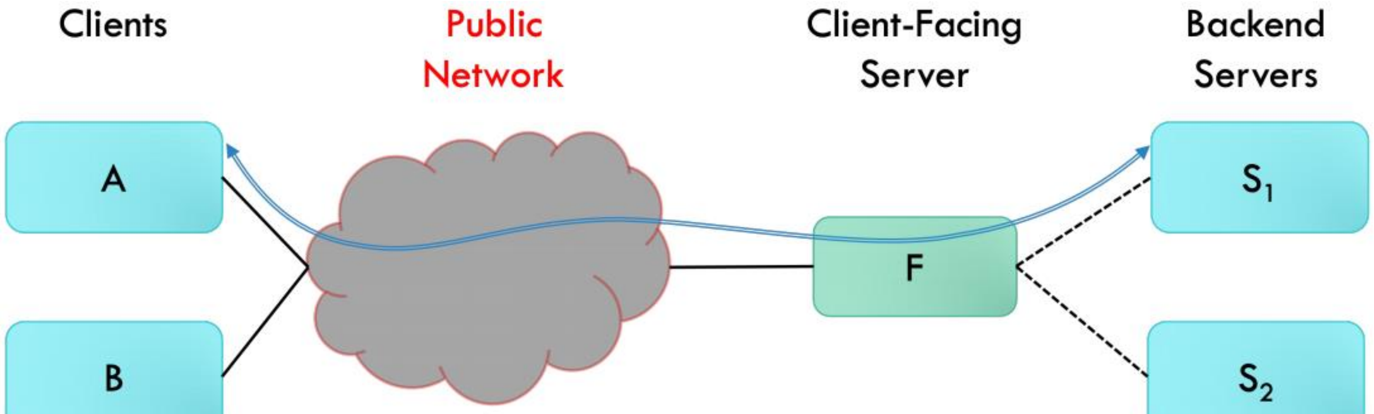
Privacy	Limitation
Client Identity Privacy (CIP)	No automated proofs Not guaranteed by TLS
Server Identity Privacy (SIP)	
Client Unlinkability (UNL)	
Client Extension Privacy (C-EXT)	
Server Extension Privacy (S-EXT)	

Encrypted Client Hello guarantees all these privacy goals

ECH (Encrypted ClientHello)

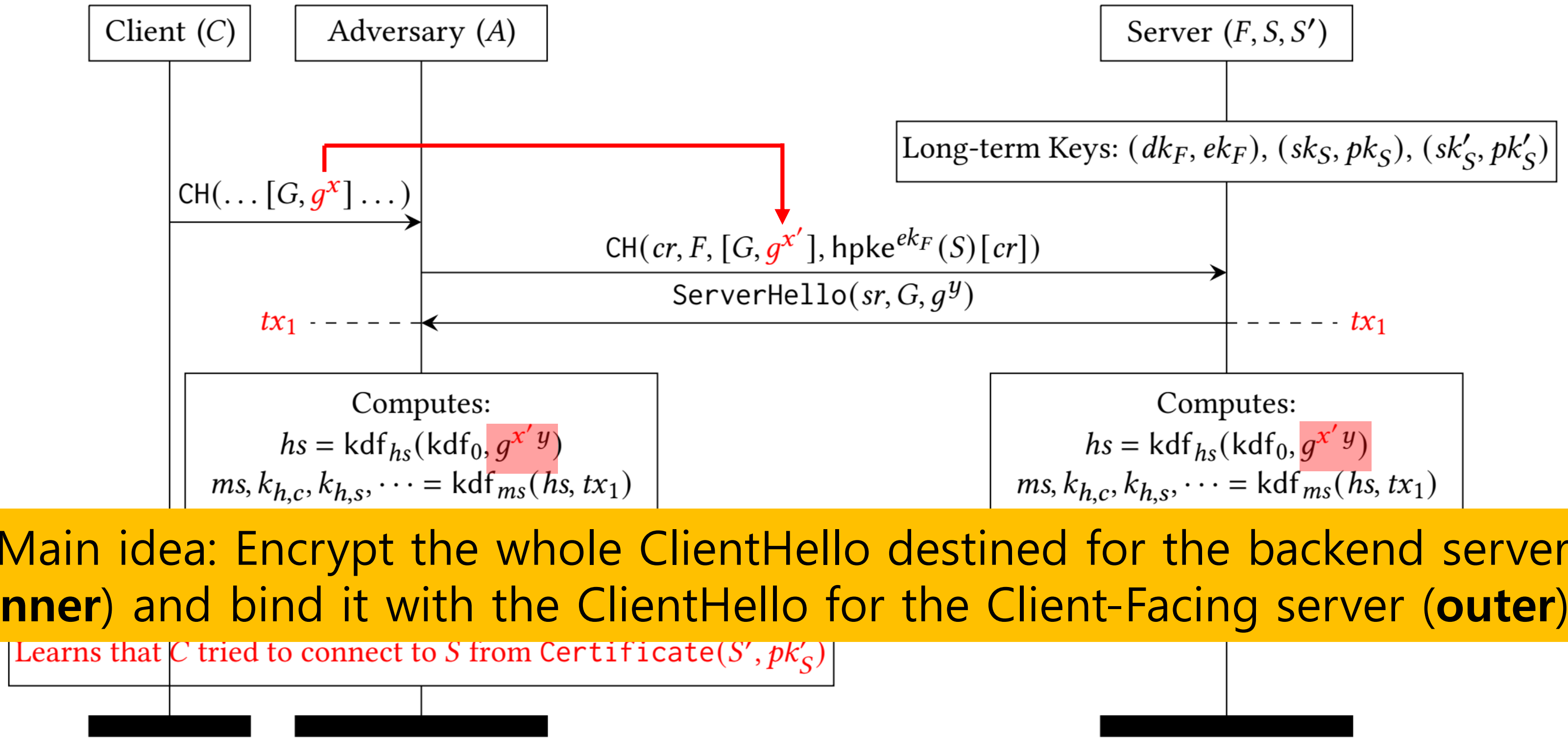
- **Goal**: Privacy guarantee of the identity of the backend server

- **Main idea**: Encrypt sensitive information (e.g., Server identity of the backend server) with a public key of the client-facing server



TLS 1.3 + ESNI (Encrypted SNI)

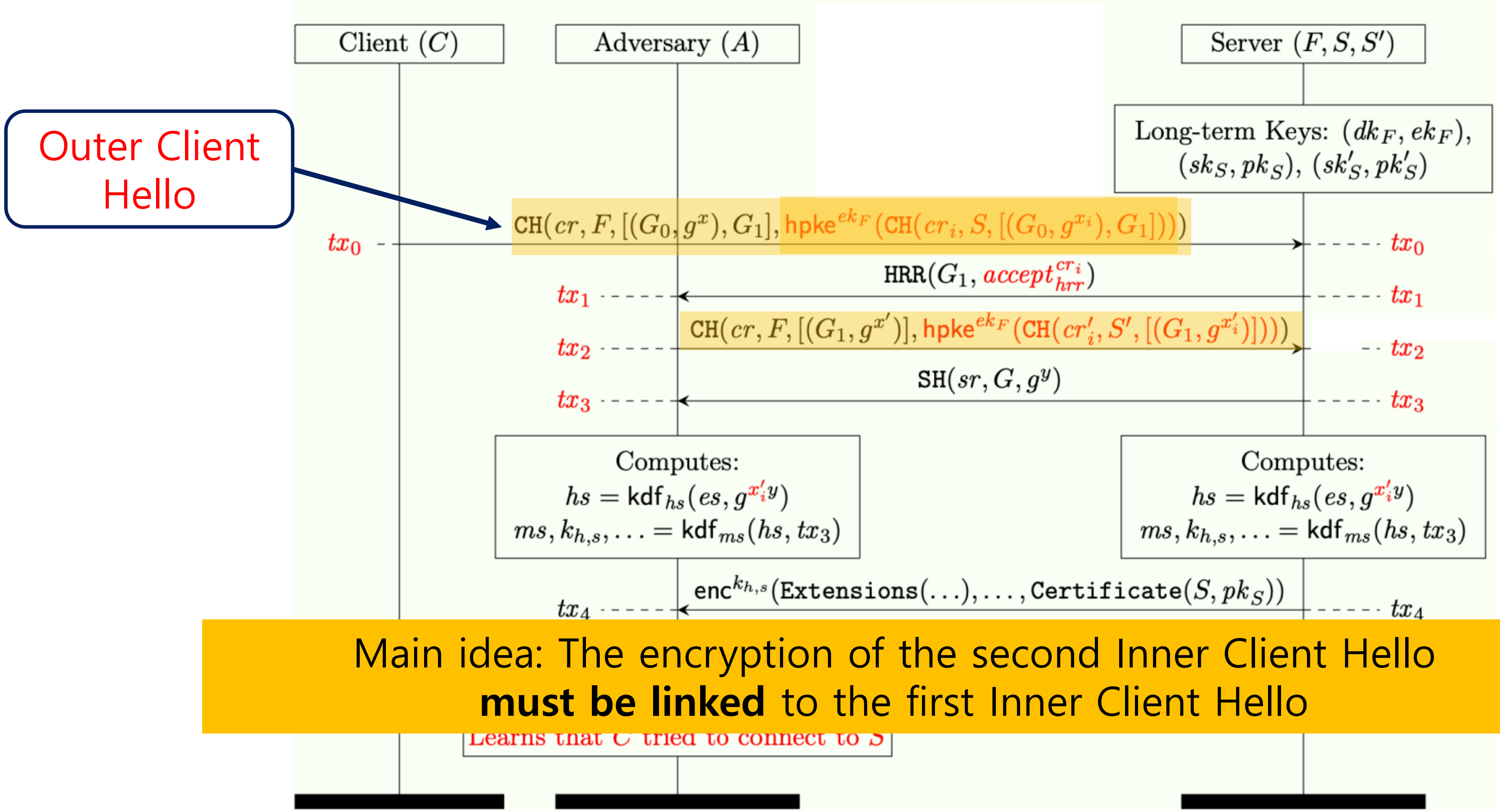
- **ESNI** is first draft of ECH
  - Encrypting the SNI extension in the ClientHello with the public-key of the client-facing server F (HPKE)
- Vulnerability



Main idea: Encrypt the whole ClientHello destined for the backend server (**inner**) and bind it with the ClientHello for the Client-Facing server (**outer**)

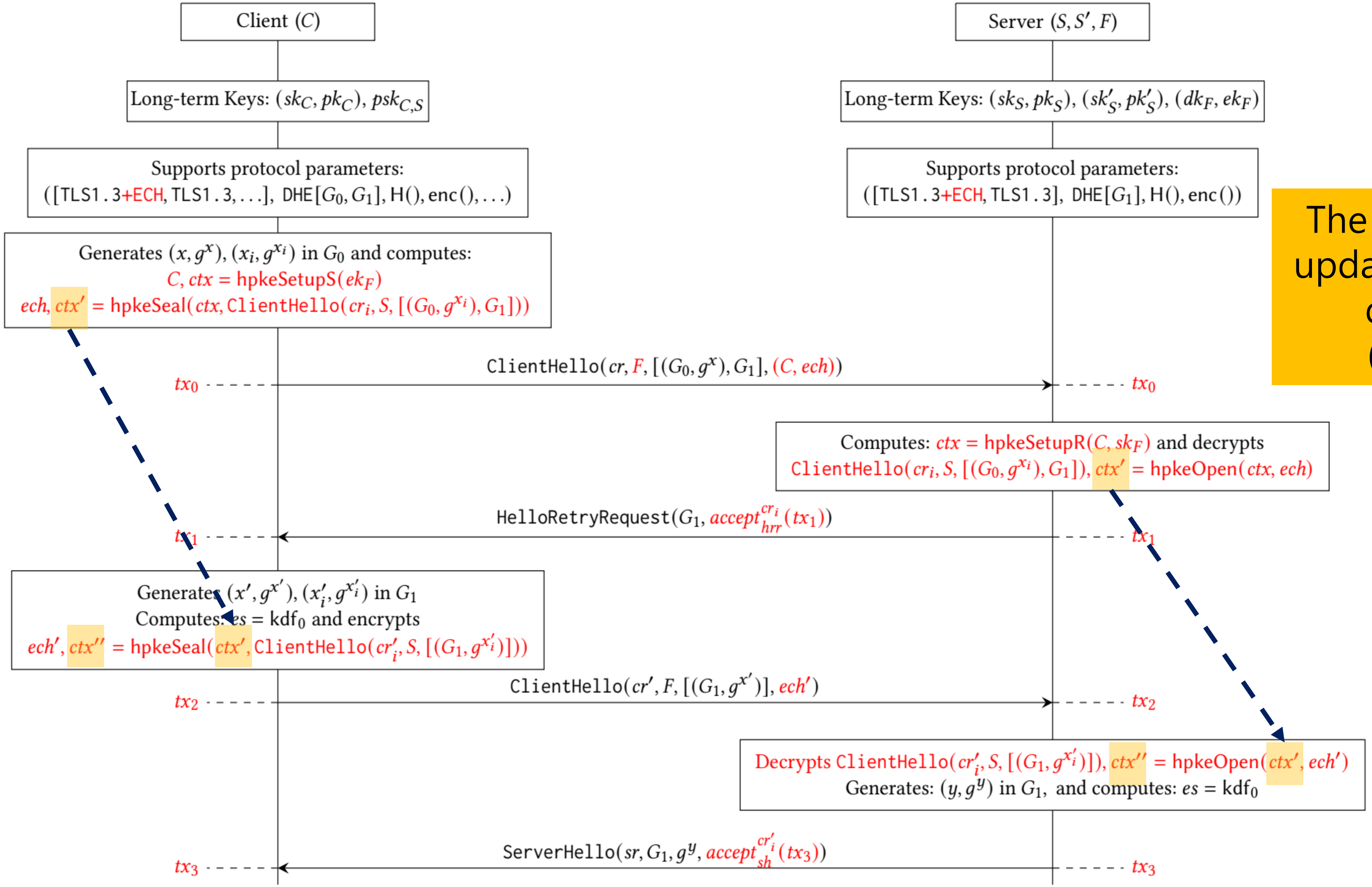
TLS 1.3 + ECH (Past)

- Another vulnerability with HelloRetryRequest



Main idea: The encryption of the second Inner Client Hello **must be linked** to the first Inner Client Hello

TLS 1.3 + ECH (Current)



The context *ctx* is updated after each decryption (ctx', ctx'')

Attacker model

- Considering the symbolic models, known as **Dolev-Yao** model
  - Attacker **can** have a control over the network, read, write and intercept messages
  - But attacker **cannot** break the cryptography nor use side channel
  - It is very powerful state-of-the-art tool

- Automated Verification Tool : **ProVerif**
  - Most common use
  - Supports multiple versions and weak ciphersuites and can find downgrade attack on TLS 1.3

Modeling

- Model Considerations
  - Focus only on TLS 1.3 (No version negotiation)
  - Model all features (e.g., HRR, PHA, PSK, Ticket, ECH, 1RTT and 0RTT Data)
  - Model all security properties (i.e., Authentication, Integrity, Confidentiality and Privacy goals)
- Proving all properties with all features is too taxing on ProVerif in computation time or memory consumption
  - OOT = **48H** and OOM = **100GB**
- **Parametrized model**: Simple configuration file allows us to activate/deactivate
  - Features
  - Compromised keys
  - Server and client behavior

Results (Authentication, Integrity, Confidentiality)

Sanity checks

Computation time

	Property	1RTT	HRR	CC	PHA	PSK-DHE	TKT	0RTT	Time
TLS1.3	All	✓	✓	✓	✓	✓	✓	✓	10h7m
TLS1.3 + ECH	SEC, UNIQ	✓	✓	✓	✓	✓	✓	✗	2h48m
	SEC0	✓	✓	✗	✓	✓	✓	✓	55m
	FS, INT	✓	✗	✓	✗	✓	✓	✗	3h40m
	CAUTH	✓	✗	✗	✓	✓	✓	✓	2h39m
		✓	✓	✓	✗	✓	✓	✗	3h26m
	SAUTH, AGR	✓	✓	✓	✗	✓	✓	✗	3h26m
	DOWN	✓	✓	✗	✗	✓	✓	✗	34h16m

✓: Feature enabled ✗: Feature disabled

Results (Privacy)

	Property	HRR	CC	PHA	PSK-DHE	TKT	Time
TLS1.3	IND, CIP UNL, S-EXT	✓	✓	✗	✓	✓	17H15
	CIP,UNL	✓	✓	✓	✓	✗	10h10m
TLS1.3 + ECH	IND	✗	✓	✗	✓	✓	21h16m
		✓	✗	✗	✗	✓	12h47
	SIP	✗	✗	✗	✓	✓	24h27m
		✓	✗	✗	✗	✗	1h13m
	CIP, UNL	✗	✓	✗	✓	✗	21h42m
		✗	✗	✗	✓	✓	35h22m
		✗	✓	✓	✗	✗	3h27m
	S-EXT,C-EXT	✓	✓	✗	✓	✗	21h20m

1-RTT and 0-RTT are disabled

Privacy properties requires more time and memory

✓: Feature enabled ✗: Feature disabled

Conclusion

- TLS 1.3 and ECH have been developed for security goals
  - The absence of verification model make hard to know that this extension works
- This paper takes first step of the automated analysis of privacy properties for the TLS 1.3 handshake
- But the limitation is still remained
  - Deactivate many of the features to try to obtain the proof in privacy
- Ongoing work: Improve ProVerif to reduce memory consumption

Thank you for listening