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

Karthikeyan Bhargavan
Inria Paris
Paris, France
karthikeyan.bhargavan@inria.fr

Vincent Cheval
Inria Paris
Paris, France
vincent.cheval@inria.fr

Christopher Wood
Cloudflare
San Francisco, United States
chriswood@cloudflare.com

ABSTRACT

TLS 1.3, the newest version of the Transport Layer Security (TLS) protocol, provides strong authentication and confidentiality guarantees that have been comprehensively analyzed in a variety of formal models. However, despite its controversial use of handshake meta-data encryption, the privacy guarantees of TLS 1.3 remain weak and poorly understood. For example, the protocol reveals the identity of the target server to network attackers, 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, previously called ESNI), but the absence of a formal privacy model makes it hard to verify that this extension works. Indeed, several early drafts of ECH were found to be vulnerable to active network attacks.

In this paper, we present the first mechanized formal analysis of privacy properties for the TLS 1.3 handshake. We study all standard modes of TLS 1.3, with and without ECH, using the symbolic protocol analyzer ProVerif. We discuss attacks on ECH, some found during the course of this study, and show how they are accounted for in the latest version. Our analysis has helped guide the standardization process for ECH and we provide concrete privacy recommendations for TLS implementors. We also contribute the most comprehensive model of TLS 1.3 to date, which can be used by designers experimenting with new extensions to the protocol. Ours is one of the largest privacy proofs attempted using an automated verification tool and may be of general interest to protocol analysts.

CCS CONCEPTS

• **Security and privacy** → **Logic and verification**; **Formal security models**; **Security protocols**; **Authentication**; **Privacy-preserving protocols**; **Pseudonymity, anonymity and untraceability**; **Web application security**.

KEYWORDS

Privacy, Protocol Security, Formal analysis, Verification, TLS, ECH

ACM Reference Format:
Karthikeyan Bhargavan, Vincent Cheval, and Christopher Wood. 2022. A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3548606.3559360

1 INTRODUCTION

The Transport Layer Security (TLS) protocol is a widely-deployed Internet standard for establishing secure channels across untrusted networks. Notably, it is used for HTTPS connections between web browsers and servers. A typical deployment scenario is depicted in Figure 1: a web browser A connects to a website S_1 across the Internet. In practice, S_1 is often run by a hosting service or content-delivery network that provides a shared client-facing server F for multiple websites. The goal of TLS is to provide a secure channel between A and S_1, even if the attacker fully controls the network, other clients like B, and other servers like S_2. Specifically, the protocol aims to provide three guarantees: *authentication* for the server S_1 (and optionally for the client A), as well as *confidentiality* and *integrity* for data exchanged between the client and server.

In 2018, the Internet Engineering Task Force (IETF) standardized TLS 1.3 [56], which improves on the previous version (TLS 1.2) in several significant ways. In terms of efficiency, TLS 1.3 reduces the latency of connection setup (called the *handshake*) from two round-trips to one round-trip before the client can start sending application data. In terms of security, the protocol provides forward secrecy as default, deprecates obsolete cryptographic constructions [4, 19, 20, 52, 59], and improves transcript authentication to prevent downgrade and key synchronization attacks [3, 6, 13, 18].

Perhaps most remarkably, TLS 1.3 was designed in collaboration with the academic research community with the explicit goal of having formal security proofs for the protocol *before* standardization. Consequently, a series of publications have analyzed the security of TLS 1.3 using a variety of proof techniques and formal definitions. A real-world protocol standard like TLS supports dozens of protocol flows depending on how the client and server are configured and analyzing all these flows by hand can be tedious. As a result, pen-and-paper cryptographic proofs tend to focus on a small subset of features supported by the protocol [15, 23, 38, 41, 47, 49, 50]. More comprehensive analyses of TLS 1.3 rely on semi-automated verification tools like Tamarin [30, 31], ProVerif [14], CryptoVerif [14], and F* [34]. All of these works focus on the three primary secure channel goals of TLS; no prior work, with the notable exception of [5], studies the *privacy* guarantees of TLS 1.3.

Privacy Goals for TLS. To provide authentication, secure channel protocols need to exchange the identities of one or both parties during the connection setup (or *handshake*). Many protocols seek to hide these identities from passive or active network adversaries. The SIGMA family of key exchange protocols [48], which inspired

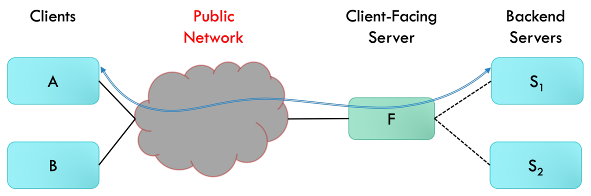

Figure 1: Example TLS 1.3 Deployment Scenario: two clients A and B (e.g. web browsers) connect via a public network to servers S_1 and S_2 (e.g. websites) that are both hosted by the same client-facing server F (e.g. a content delivery network).

both IKEv2 [45] and TLS, includes two identity hiding variants. The Noise framework defines multiple protocols with different levels of identity privacy [54], including the one used in WireGuard [37].

In the TLS scenario of Figure 1, we can state two identity privacy goals. *Client identity privacy* says that the attacker cannot distinguish between an authenticated connection from A to S_1 and one from B to S_1. *Server identity privacy* says the attacker cannot distinguish between a connection from A to S_1 and one from A to S_2. In addition, we may also want to protect other sensitive handshake metadata. We can then ask whether these privacy guarantees hold for arbitrary sequences of connections between honest participants, in the presence of passive or active network adversaries.

Before TLS 1.2, the identities of the client and server were sent in the clear and hence these privacy goals were trivially false. In TLS 1.3, however, most of the handshake messages, including those containing client and server certificates are encrypted. This handshake encryption feature has raised some controversy, since it prevents legacy network middleboxes from being able to inspect and filter TLS connections [33, 44, 58], but it is still included in TLS 1.3 to guarantee better privacy against censorship and surveillance [11].

Despite this goal, TLS 1.3 still leaks the server identity to passive network attackers. In the first handshake message (`ClientHello`), A sends the name of S_1 in the clear so that the client-facing server F can route the connection to the right server. Even client identity privacy can be quite subtle, since a network adversary may be able to correlate a sequence of connections made by the same client to the same server, if these connections use the same pre-shared key.

Encrypted Client Hello. To improve the privacy of TLS 1.3, the IETF TLS working group is standardizing a new protocol extension called Encrypted Client Hello (ECH) [57], which was previously called Encrypted Server Name Indication (ESNI). The key idea of ECH is to encrypt parts of the `ClientHello` in order to hide the server name, pre-shared key, and other handshake metadata.

The main goal of ECH is to provide server identity privacy against active network attackers. However many early designs for ESNI and ECH proved to be vulnerable to subtle attacks, some of which we shall detail later in this paper. The TLS working group invited researchers to formally analyze the ECH design before standardization and this work is the first to provide such an analysis.

Safely Extending TLS 1.3. ECH is the first major extension to TLS 1.3 since it was standardized. It introduces new cryptographic constructions to the handshake and subtly changes the meaning of the handshake transcript. Given that TLS 1.3 has seen such extensive security analysis, it is a valid concern that the changes introduced by ECH may break the security guarantees of TLS 1.3.

We advocate a three-step analysis methodology for all new security and privacy extensions to TLS 1.3:

(1) **Security Preservation**: Prove that the extended protocol preserves the authentication, confidentiality, and integrity properties that have been previously proved for TLS 1.3;
(2) **Stronger Guarantees:** Define new security goals for TLS; show that vanilla TLS 1.3 does not satisfy these goals and prove that the extended protocol does;
(3) **Downgrade Resistance:** Prove that an active network attacker cannot downgrade the extended protocol to vanilla TLS 1.3, hence removing the extended guarantees.

Furthermore, we advocate that this analysis should be applied to a model of TLS 1.3 that covers as many of its modes and optional features as possible, to ensure that we do not miss attacks that only appear in specific configurations of the protocol.

Our Approach and Contributions. We begin with the ProVerif model of TLS 1.3 in [14] and make it more precise and more comprehensive: we carefully model all the configuration options and resulting protocol branches, we add support for `HelloRetryRequest` and post-handshake authentication, and we define stronger confidentiality and integrity goals. We believe that the result of these extensions is the most detailed formal model of TLS 1.3 to date, which may be of independent interest to protocol designers and analysts. We prove, using ProVerif, that this model satisfies the classic secure channel goals of TLS 1.3. We note that ProVerif is a symbolic prover based on the Dolev-Yao model [36], so proofs in ProVerif are not directly comparable with cryptographic pen-and-paper proofs.

Next, we formally define a series of privacy goals for TLS 1.3, including client identity privacy (for authenticated clients), client unlinkability (for anonymous clients), server identity privacy, and metadata privacy for additional extensions sent by the client and server. We prove that vanilla TLS 1.3 satisfies some of these properties. Our formulation of privacy for TLS 1.3 is new, and our proofs are the first to be machine-checked. The main prior work in this space is [5], which presents a pen-and-paper proof of client unlinkability (without certificates) and a limited form of server identity privacy for two protocol flows of TLS 1.3. Our results are for more privacy properties and more protocol flows, albeit using the more abstract Dolev-Yao model of cryptography.

Finally, we extend our model with ECH and prove that TLS 1.3 with ECH preserves the security properties of TLS 1.3, that it achieves the privacy goals, and that it protects against downgrades to TLS 1.3. We also show that previous draft versions of ECH did not satisfy the privacy goals and demonstrate attacks on these versions.

Our analysis provides concrete feedback for the TLS ECH proposal for the TLS working group and our results have already influenced the standardization of ECH. We also provide guidelines for implementors on how to implement TLS 1.3 and ECH safely.

Our proofs of privacy are (to our knowledge) the largest automated privacy proofs for any protocol in the literature, and at the cutting edge of what is achievable by modern verification tools.

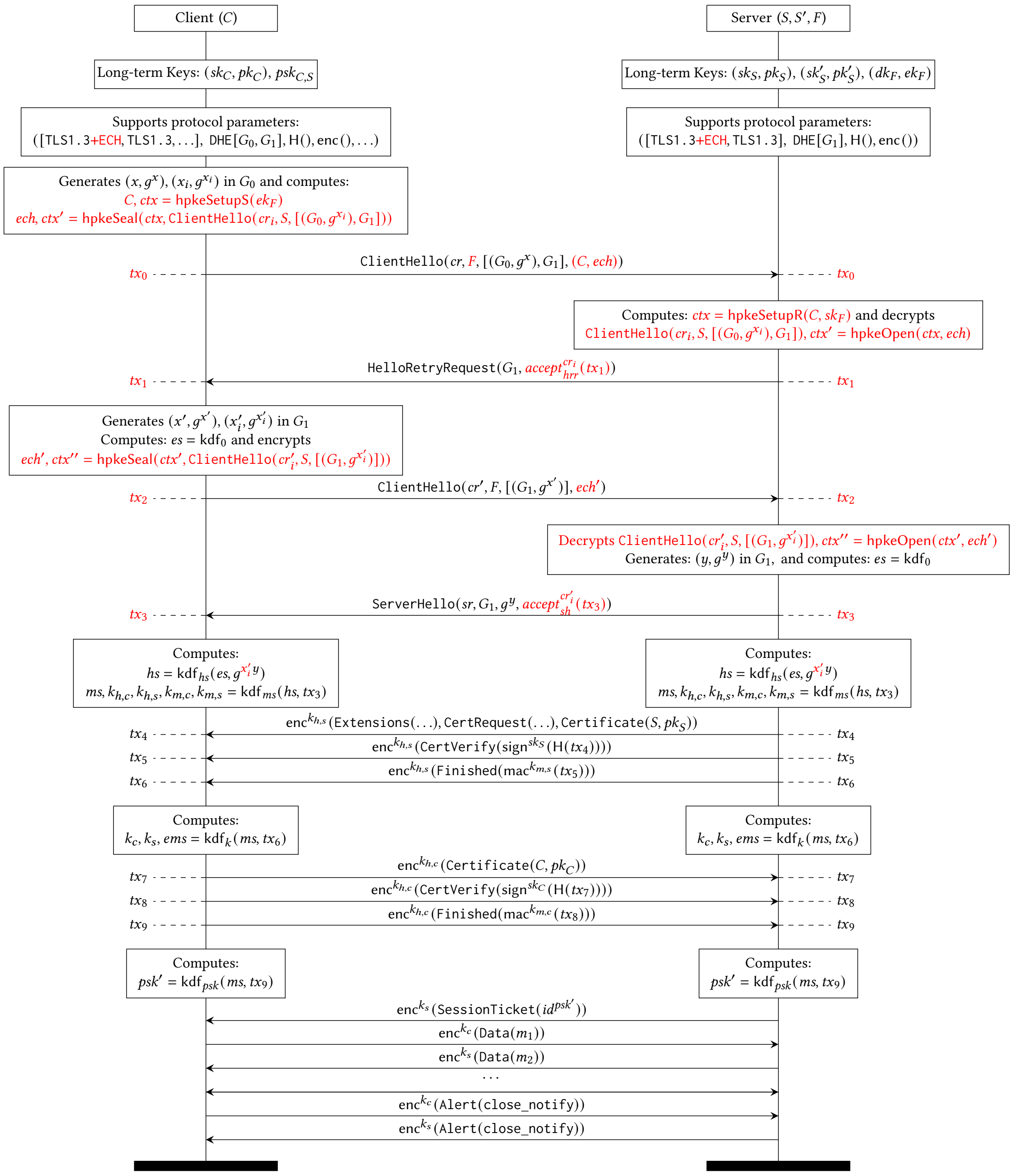
Client (C)
Server (S, S', F)
Long-term Keys: (sk_C, pk_C), $psk_{C,S}$
Long-term Keys: (sk_S, pk_S), (sk'_S, pk'_S), (dk_F, ek_F)
Supports protocol parameters: ([TLS1.3+ECH, TLS1.3, ...], DHE[G_0, G_1], H(), enc(), ...)
Supports protocol parameters: ([TLS1.3+ECH, TLS1.3], DHE[G_1], H(), enc())
Generates (x, g^x), (x_i, g^{x_i}) in G_0 and computes: C, ctx = hpkeSetupS(ek_F) ech, ctx' = hpkeSeal(ctx, ClientHello($cr_i, S, [(G_0, g^{x_i}), G_1]$))
tx_0 ClientHello($cr, F, [(G_0, g^x), G_1], (C, ech)$) tx_0
Computes: ctx = hpkeSetupR(C, sk_F) and decrypts ClientHello($cr_i, S, [(G_0, g^{x_i}), G_1]$), ctx' = hpkeOpen(ctx, ech)
tx_1 HelloRetryRequest(G_1, $accept^{cr_i}_{hrr}(tx_1)$) tx_1
Generates $(x', g^{x'})$, $(x'_i, g^{x'_i})$ in G_1 Computes: es = kdf$_0$ and encrypts ech', ctx'' = hpkeSeal(ctx', ClientHello($cr'_i, S, [(G_1, g^{x'_i})]$))
tx_2 ClientHello($cr', F, [(G_1, g^{x'})], ech'$) tx_2
Decrypts ClientHello($cr'_i, S, [(G_1, g^{x'_i})]$), ctx'' = hpkeOpen(ctx', ech') Generates: (y, g^y) in G_1, and computes: es = kdf$_0$
tx_3 ServerHello(sr, G_1, g^y, $accept^{cr'_i}_{sh}(tx_3)$) tx_3
Computes: hs = kdf$_{hs}(es, g^{x'_i y})$ $ms, k_{h,c}, k_{h,s}, k_{m,c}, k_{m,s}$ = kdf$_{ms}(hs, tx_3)$
Computes: hs = kdf$_{hs}(es, g^{x'_i y})$ $ms, k_{h,c}, k_{h,s}, k_{m,c}, k_{m,s}$ = kdf$_{ms}(hs, tx_3)$
tx_4 enc$^{k_{h,s}}$(Extensions(...), CertRequest(...), Certificate(S, pk_S)) tx_4
tx_5 enc$^{k_{h,s}}$(CertVerify(sign^{sk_S}(H(tx_4)))) tx_5
tx_6 enc$^{k_{h,s}}$(Finished(mac$^{k_{m,s}}$(tx_5))) tx_6
Computes: k_c, k_s, ems = kdf$_k(ms, tx_6)$
Computes: k_c, k_s, ems = kdf$_k(ms, tx_6)$
tx_7 enc$^{k_{h,c}}$(Certificate(C, pk_C)) tx_7
tx_8 enc$^{k_{h,c}}$(CertVerify(sign^{sk_C}(H(tx_7)))) tx_8
tx_9 enc$^{k_{h,c}}$(Finished(mac$^{k_{m,c}}$(tx_8))) tx_9
Computes: psk' = kdf$_{psk}(ms, tx_9)$
Computes: psk' = kdf$_{psk}(ms, tx_9)$
enc^{k_s}(SessionTicket($id^{psk'}$))
enc^{k_c}(Data(m_1))
enc^{k_s}(Data(m_2))
...
enc^{k_c}(Alert(close_notify))
enc^{k_s}(Alert(close_notify))

Figure 2: TLS 1.3 Protocol Flow for (EC)DHE Handshake with Encrypted Client Hello (ECH) and 1-RTT Data. This protocol flow uses a Diffie-Hellman key exchange with certificate-based server authentication. It also shows `HelloRetryRequest`-based group negotiation and certificate-based client authentication. Other optional features like version and ciphersuite negotiation, post-handshake client authentication and PSK-based mutual authentication are not depicted. Message components in red are introduced by the ECH extension: `ClientHello` messages now have an inner `ClientHello` encrypted with the HPKE public-key pk_F of the front-end server (F). Encryption of `ClientHello` using SealS **is bound to the outer `ClientHello` as described in the specification, though omitted here for clarity. If the server accepts ECH, then the inner `ClientHello` is used in the transcripts ($tx_0, tx_2, \ldots$), and the inner key-share ($g^{x'_i}$) is used for Diffie-Hellman. Otherwise, the connection falls back to standard TLS 1.3 and the outer `ClientHello` and its key-share ($g^{x'}$) are used.**

2 TLS 1.3 PROTOCOL DESIGN AND FEATURES

TLS 1.3 is specified in IETF RFC 8446 [56] as a two-party secure channel protocol between a client and a server. It composes three sub-protocols: the *handshake* sub-protocol performs an authenticated key exchange (AKE) to establish a fresh symmetric key, the *record* protocol uses this key for authenticated encryption of application data (and handshake messages), and the *alert* sub-protocol is used to indicate connection closure and fatal errors.

Figure 2 shows an example flow of messages in a TLS 1.3 connection. For now, we ignore the parts in red, which will be used to explain the ECH extension in Section 4. Many of the details of TLS 1.3 have been extensively covered and analyzed in prior work. Here, we focus on the features that are most relevant for the privacy modeling and analysis in this paper. We begin with the standard TLS 1.3 handshake mode, and then describe various optional features.

Server Authenticated (EC)DHE Handshake with 1-RTT Data. When a TLS 1.3 client (C) connects to a server (S) for the first time, it typically runs a Diffie-Hellman key exchange (using an elliptic curve) where the server is authenticated with an X.509 public key certificate but the client is unauthenticated.

The protocol begins when C sends a `ClientHello` message containing a nonce (cr) and a Diffie-Hellman key share (g^x) in some group or elliptic curve (G_0). The server responds with a `ServerHello` containing its own nonce (sr) and key share (g^y). The client and server then both compute the Diffie-Hellman shared secret (g^{xy}) and use it to derive a pair of handshake encryption keys ($k_{h,c}, k_{h,s}$) and a pair of MAC keys ($k_{m,c}, k_{m,s}$).

The server then sends a sequence of four handshake messages (encrypted using $k_{h,s}$): `Extensions` contains additional server parameters, `Certificate` contains the server's public-key certificate (for pk_S), `CertVerify` contains a signature over the handshake transcript so far using the server's private key (sk_S), and `Finished` contains a MAC (using $k_{m,s}$) over the handshake transcript up to `CertVerify`. Here, The `Certificate` and `CertVerify` messages serve to authenticate the server, while `Finished` provides key and transcript confirmation, following the classic Sign-and-MAC protocol design pattern from SIGMA [48].

The client responds by sending its own `Finished` message (encrypted using $k_{h,c}$) containing a MAC (using $k_{m,c}$) over the handshake transcript so far (i.e. up to the server `Finished`).

At this point, both client and server derive a master secret (ms) and data encryption keys (k_c, k_s), and then switch over to these data encryption keys to exchange a stream of encrypted application data messages ($m_1, m_2, \ldots$) in both directions. Application data sent after `Finished` is typically called *1RTT Data* since it is sent after one set of handshake messages has been sent in each direction. When the data exchange is complete, the client and server send `close_notify` alerts to close the connection.

Cryptographic computations for the various key derivations and MACs in the TLS 1.3 handshake are detailed in [16, Figure 3].

Negotiating Connection Parameters. The basic handshake described above assumes that the client and server already agree on the TLS protocol version, Diffie-Hellman group, and the cryptographic algorithms they will use for signing, MAC, encryption etc. In practice, the client and server *negotiate* the values of these parameters in the beginning of the handshake.

If the client supports multiple protocol versions (e.g. TLS 1.3 and TLS 1.2), it lists these versions in the `ClientHello` and the server chooses the version it prefers (typically the highest version it supports). Similarly, the `ClientHello` lists the signature algorithms, Diffie-Hellman groups, encryption schemes, and hash algorithms that the client supports, and the server chooses the combination it prefers and indicates it in the `ServerHello`.

If the server chooses a Diffie-Hellman group G_1 that the client supports but the client has not provided a key share for G_1 in the `ClientHello`, the server sends back an `HelloRetryRequest` message with the group G_1; the client responds with a new `ClientHello` containing a key-share ($g^{x'}$) for G_1. The subsequent handshake transcript includes the full transcript with both `ClientHello` messages.

Certificate-based Client Authentication. Although server authentication is far more common, TLS 1.3 also allows clients to be authenticated using public key certificates. During the handshake, between its `Certificate` and `CertVerify` messages, the server may send a `CertRequest` message indicating that it wishes for the client to authenticate itself. If the client agrees to this request, then in its response, it sends three messages instead of just a `Finished`: `Certificate` contains the client's public-key certificate (for pk_C), `CertVerify` contains the client's signature over the transcript (using sk_C), `Finished` contains a MAC over the transcript up to the client's `CertVerify` (using $k_{m,c}$). All these messages are encrypted using the client's handshake encryption key ($k_{h,c}$).

In some scenarios, the server may wish to request client authentication after the handshake is finished. This feature is called *post-handshake authentication*: the server sends a `CertRequest` message in the middle of the 1RTT application data exchange, and the client responds with the `Certificate-CertVerify-Finished` message combination. These messages are encrypted under the client's data encryption key (k_c).

Pre-Shared Keys (PSK). If a client (C) and server (S) have been configured with a pre-shared symmetric key ($psk_{C,S}$), then they can avoid the expensive public-key signature operations and instead use this PSK to authenticate each other. This pre-shared key may be an *external PSK* provided by the application or it may be a *resumption PSK* output by a prior handshake between the client and server.

[16, Figure 4] depicts a typical PSK-DHE handshake: the client lists the identifiers (e.g. $id^{psk_{C,S}}$) of the PSKs (e.g. $psk_{C,S}$) that it shares with the server in the `ClientHello`. It proves its knowledge of each PSK by using it to compute a *binder* ($binder^{psk_{C,S}}$), i.e. a MAC (using $psk_{C,S}$) over the current transcript (including the current `ClientHello` but with binders removed).

The server responds by choosing one of the PSKs from the `ClientHello` and indicates its index in the `ServerHello`. The chosen PSK is mixed into all subsequent encryption and MAC keys. The server then sends an `Extensions` message followed directly by `Finished`. The MAC in the `Finished` serves to authenticate the server since it could only have been produced by someone who knows the PSK. Note that if the PSK is known to someone other than C or S, or if C is willing to use the PSK both as a client and server, then the guarantees of this MAC are very weak [40].

The client completes the handshake with its `Finished` message, hence authenticated itself by proving its knowledge of the PSK.

Tickets and 0RTT. After the end of each handshake, whether it uses certificates or PSKs, the server may send the client a *session ticket* (`SessionTicket`) that serves as a new PSK identifier. This message is encrypted under the application data key (k_s). The client and server then derive a fresh *resumption PSK* (psk') from the master secret (ms) and save it for use in subsequent PSK handshakes.

In a PSK handshake, the client already has a key it shares with the server, and so it can start sending *0RTT* application data immediately after the `ClientHello` message without waiting for the server to respond. This data is called *0RTT Data* and is encrypted using a key derived from the PSK. If the server does not accept 0RTT data or does not choose the PSK, then this data is discarded.

TLS Extensions. We have described the main TLS 1.3 protocol flows and commonly-used optional features, but the protocol itself is extensible. The `ClientHello` message may indicate protocol extensions that the client supports and the server may choose some of these extensions in the `ServerHello`.

A commonly used TLS extension on the Web is *Server Name Indication* (SNI): the `ClientHello` includes the name of the server (S) to which the client wishes to connect. SNI is 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`, and `HelloRetryRequest` messages are unencrypted, but the server can encrypt some extension data in its `Extensions` message. As we shall see, the ECH extension allows the client to also encrypt elements of the `ClientHello`, including the SNI extension.

3 TLS 1.3 GOALS AND FORMAL ANALYSES

In this section, we outline the main security and privacy goals of TLS 1.3 and describe various formal analyses of the protocol.

Threat Model. The threat model considered by TLS 1.3 includes passive and active network attackers and malicious or compromised clients and servers. We say that a client or server is *compromised* if any of its private keys or pre-shared keys is known to the adversary; otherwise we say that it is *honest*. The attacker can use compromised keys to impersonate a client or server to an honest party. Furthermore, the attacker is free to use any cryptographic construction to construct and decode messages.

The security assumptions on the cryptographic constructions can be formally stated in many different ways. In the *symbolic* or Dolev-Yao model [36], cryptographic functions are perfect black-boxes that obey well-defined algebraic rules; protocol participants and the attacker are modeled as non-deterministic processes that can participate in an unbounded number of sessions but they cannot violate the algebraic rules of cryptography. In the *computational* or complexity-theoretic model, cryptographic constructions are probabilistic functions over bitstrings; protocol participants and the attacker are probabilistic polynomial-time Turing machines. To better understand the relationship between these two models and their applications, see [2, 8, 28]. At a high-level, computational models are more cryptographically precise but symbolic models are easier to analyze and can scale up to large protocols with many optional features. The strengths of both models are complementary.

Authentication and Integrity Goals. The main authentication and integrity goals of TLS 1.3 can be stated informally as follows:

- **Server Authentication** (SAUTH): If a client C receives application data message m over a TLS 1.3 connection (supposedly) with a server S, then either there must be an honest server S that sent the message m, or else (a long-term key of) S must be compromised.
- **Client Authentication** (CAUTH): If a server S receives application data m over a client-authenticated TLS 1.3 connection (supposedly) with a client C, then either there must be an honest client C that sent the message m, or else (a long-term key of) C must be compromised.
- **Key and Transcript Agreement** (AGR): If a client (or server) completes a handshake with an honest peer, then both parties must agree on the connection keys (e.g. k_c, k_s) and the handshake transcript.
- **Data Stream Integrity** (INT): If a client (or server) receives a sequence of messages $m_0, m_i, \ldots$ over a TLS 1.3 connection, then either an honest peer sent the same sequence of messages, or else the peer must be compromised.
- **Key Uniqueness** (UNIQ): Two different TLS 1.3 connections cannot result in the same encryption keys.
- **Downgrade Resilience** (DOWN): If a client (or server) completes a handshake with an honest peer and both parties prefer to use a certain protocol parameter (e.g a version or ciphersuite or Diffie-Hellman group), then the handshake cannot end with the less-preferred parameter.

Note that the authentication properties (CAUTH, SAUTH) apply to both certificate-based handshakes and PSK handshakes. In the case of certificate-based authentication, these properties forbid key compromise impersonation (KCI): even if an attacker knows (say) sk_C, it cannot impersonate S to C. The transcript agreement property (AGR) provides different guarantees at the client and server: after the client `Finished`, the server gets full transcript agreement, whereas the client only gets transcript agreement up to the server `Finished`. Data stream integrity (INT) implies that the attacker cannot reorder or selectively drop application data messages. Key uniqueness (UNIQ) prevents unknown key share and key synchronization attacks [18]. Downgrade resilience (DOWN) prevents a network attacker from fooling modern clients and servers into using an obsolete protocol version or cryptographic construction that is supported only for backwards compatibility.

Confidentiality. The confidentiality guarantees of TLS 1.3 can either be stated in terms of the keys established by the handshake or the application data exchanged by the client and server:

- **Key Secrecy** (SEC): If a client (or server) completes a handshake with data encryption keys (k_c, k_s), then either the peer is honest and keys are unknown to the adversary, or the peer is compromised.
- **Key Indistinguishability** (IND): If a client (or server) completes a handshake with an honest peer to obtain a new key (e.g. psk'), then this key is indistinguishable from a fresh random value generated at the end of the handshake.

	(EC)DHE + Server Auth		Negotiation			Client Auth		Pre-Shared Keys				Extensions	
Method	Handshake	1RTT	VER	CS	HRR	CC	PHA	PSK-DHE	TKT	PSKO	0RTT	SNI	ECH
Pen-and-Paper [39]	✓	✗	✗	✗	✗	✓	✗	✓	✗	✓	✗	✗	✗
Pen-and-Paper [5]	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗
CryptoVerif [14, 21]	✓	✓	✗	✗	✗	✓	✗	✓	✓	✓	✓	✗	✗
F* [17, 35]	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
Tamarin [30, 31]	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✗	✗
ProVerif [14]	✓	✓	✓	✓	✓	✓	✗	✓	✗	✗	✓	✗	✗
This work (ProVerif)	✓	✓	✗	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓

		Authentication						Confidentiality				Privacy			
Method	Model	SAUTH	CAUTH	AGR	INT	UNIQ	DOWN	SEC	FS	IND	SEC0	CIP	UNL	SIP	EXT
Pen-and-Paper [39]	Comp.	✓	✓	✓	✗	✓	✗	✓	✓	✓	✗	✗	✗	✗	✗
Pen-and-Paper [5]	Comp.	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗
CryptoVerif [14, 21]	Comp.	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗
F* [17, 35]	Comp.	✗	✗	✗	✓	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗
Tamarin [30, 31]	Symb.	✓	✓	✓	✓	✓	✗	✓	✓	✗	✓	✗	✗	✗	✗
ProVerif [14]	Symb.	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗
This work (ProVerif)	Symb.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: TLS 1.3 Features (top) and Security Properties (bottom) covered by various formal analyses.
Handshake: (EC)DHE handshake; 1RTT: 1RTT Data; VER: multiple TLS versions; CS: multiple ciphersuites; HRR: group negotiation; CC: client certificates; PHA: post-handshake authentication; PSK-DHE: PSK-(EC)DHE handshake; TKT: session tickets; PSKO: psk-only mode; 0RTT: 0RTT Data; SNI: server name indication; ECH: encrypted client hello

- **1RTT Data Forward Secrecy** (FS): If a client (or server) sends a secret 1RTT data message *m* over a TLS 1.3 connection with an honest peer, then *m* remains secret even if the peer is compromised after the connection has closed.
- **0RTT Data Secrecy** (SEC0): If a client sends secret 0RTT application data message *m* over a TLS 1.3 connection, then this message remains secret unless the PSK used in the connection is compromised.

In the above, indistinguishability (IND) is a strictly stronger property than (syntactic) secrecy (SEC) and forward secrecy for 1RTT (FS) is also stronger than 0RTT secrecy (SEC0). These are four example properties taken from the literature on TLS 1.3, but other confidentiality properties, such as data indistinguishability and post-compromise secrecy, can also be formulated.

Privacy. Appendix E.1 of the TLS 1.3 RFC [56] mentions the "protection of endpoint identities" as one of the goals of the handshake. We elaborate on this to state four desired privacy goals:

- **Client Identity Privacy** (CIP): If one of the two clients A and B connects using client-authenticated TLS 1.3 to an honest server S, then the attacker cannot tell which of the two clients made the connection.
- **Client Unlinkability** (UNL): If both A and B make TLS 1.3 connections to an honest server S, and then one of them makes a new TLS 1.3 connection to the same server S, then the attacker cannot tell which of the two clients made the new connection.
- **Server Identity Privacy** (SIP): If an honest server F holds long-term keys for two servers S_1 and S_2 and a client connects to F, then the attacker cannot tell whether the connection uses the server identity S_1 or S_2.
- **Client and Server Extension Privacy** (EXT): If a client or server sends an extension with a sensitive payload over a TLS 1.3 handshake with an honest peer, then the attacker cannot distinguish between different values of the payload.

Note that client identity privacy (CIP) is stated for both certificate-based and PSK-based client authentication. Client unlinkability (UNL) is stronger in that it works even for unauthenticated clients that make a series of connections that are cryptographically linked via resumption PSKs. The TLS 1.3 RFC only asks for Server identity privacy (SIP) against passive attackers, and our formulation is stronger. Extension privacy (EXT) is stated as indistinguishability for extension payloads, and is treated here as a privacy property since extensions typically carry connection metadata.

Each privacy goal must hold even for partial handshakes that may eventually fail, unlike secrecy and authentication goals, which usually apply only to successful connections. However, each goal is only expected to hold under some reasonable assumptions, without which there would be trivial attacks. For example, for Client Unlinkability, we need to assume that either both connections ($A - S$ and $B - S$) use a PSK or neither of them do; otherwise, the attacker can trivially distinguish between the two, since the server sends fewer messages in PSK handshakes. In Section 5, we document all our assumptions for each privacy goal.

Although our authenticity, confidentiality, and privacy goals are informally described here in terms of a few specific clients (A, B) and servers (S_1, S_2), these goals are expected to hold in scenarios where the target sessions run in parallel with an unbounded number of sessions between honest or compromised clients and servers (see Section 5.2 for more modelling details).

Prior Security Analyses. Many prior works analyze various modes of TLS 1.3 for authentication and confidentiality properties. We survey a few of the most advanced works in this space and compare the TLS 1.3 features and security properties they cover in Table 1.

Pen-and-paper cryptographic proofs of TLS 1.3 typically focus on specific handshake modes to highlight and analyze a particular feature of the protocol [15, 23, 38, 41, 47, 49, 50]. The most recent cryptographic proof of the TLS 1.3 handshake appears in [39] and subsumes many prior analyses. It covers the certificate-based DHE handshake, the PSK-DHE handshake, and the PSK-Only handshake. However, it does not cover data or handshake encryption, negotiation, session tickets, or extensions. It proves confidentiality and integrity properties about handshake keys, but does not prove properties about application data (e.g. integrity) or about privacy.

Suppose there were an attack that exploited a combination of `HelloRetryRequest` and session tickets to only manifest in a sequence of three handshakes, where the resumption PSK generated in the first handshake is used in both subsequent handshakes. Such complex attacks have traditionally appeared in TLS (see e.g. [18]), but are well outside the scope of [39]. Even with these limitations, this is an impressive proof described carefully in 65 pages of text.

The problem is that pen-and-paper proofs cannot scale to the full complexity of real-world protocols like TLS. Even if we could write hundreds of pages of proofs covering all the optional features, checking them would be infeasible. One option is to build machine-checked cryptographic proofs, using tools like CryptoVerif [14, 21]. This approach covers more TLS features than pen-and-paper analyses but the proofs have to be guided manually and do not easily scale to the analysis of (say) extensions or group negotiation.

The most comprehensive analyses of TLS rely on automated tools based on the symbolic Dolev-Yao model. For example, Tamarin has been used to develop a comprehensive model of most features of TLS 1.3 [30, 31]. This model, however, does not cover version and ciphersuite negotiation, probably to reduce the analysis complexity. A different symbolic model was developed in ProVerif [14], which supports multiple versions and weak ciphersuites and hence can find downgrade attacks on TLS 1.3. However, neither of these models analyzes stronger properties like key indistinguishability (IND) or privacy. The probable reason is that these properties require significant modeling effort, precision, and computation time.

A different angle of research is to analyze the security of TLS 1.3 *implementations*, to ensure that the proof applies to the deployed protocol without modeling abstractions. This approach has resulted in proofs for the TLS 1.3 and QUIC record layers in F* [17, 35], but it has not yet been applied to the handshake.

Most prior works on TLS 1.3 do not consider privacy. A notable exception is [5] which describes several privacy attacks on TLS 1.3, and gives a pen-and-paper proof of server identity privacy and client unlinkability for the DHE, PSK-DHE, and PSK-Only handshakes. The model does not cover client authentication and does not enable SNI (which would falsify server identity privacy). This work serves to emphasize the weak privacy guarantees of TLS 1.3. In order to get stronger privacy, a new extension to TLS 1.3 is needed, and to comprehensively analyze such an extension, we advocate using a machine-checked symbolic model of TLS 1.3.

Our analysis results on TLS 1.3. In this paper, we present a new configurable model of TLS 1.3 in ProVerif. Our model extends the previous Proverif model of [14] in several ways: we add support for post-handshake authentication and ticket-based resumption, we add extensions like SNI and ECH, and we make the model significantly more configurable, so that the analyst can easily enable and disable various features and rerun ProVerif. All the analyses in this paper are run using automated scripts that verify the protocol under various combinations of features, until they find the maximal set of features under which ProVerif is able to terminate in 48 hours.

Another advance we make over the TLS 1.3 model of [14] is to add support for equivalence-based reasoning. We prove key indistinguishability for TLS 1.3, stated as the indistinguishability of the resumption master secret from a random value. This kind of *strong secrecy* property was not proved before for TLS 1.3 in ProVerif or in Tamarin. We also analyze TLS 1.3 for privacy properties, finding that TLS 1.3 with SNI immediately fails server identity privacy (SIP), does not provide client extension privacy (C-EXT), but does provide client identity privacy (CIP), unlinkability (UNL), and server extension privacy (S-EXT).

Conversely, to keep the analysis feasible, we do not support version and ciphersuite negotiation, and hence do not model downgrade attacks to prior versions and legacy cryptographic algorithms. We also do not support PSK-Only handshakes (without ECDHE). The full list of features supported in our model is depicted in Table 1.

Our verification results for TLS 1.3 are depicted in the top rows of Table 2. Reachability properties like authentication and secrecy were fully verified without any restrictions. For equivalence-based properties like indistinguishability and client identity privacy, the analysis did not terminate (under 48 hours) with all features enabled. Hence, we used the configurability of our model to analyze 621 combinations of features to find the best combinations where ProVerif gave us a result. These choices are discussed in Section 5.

Our model of TLS 1.3 is one of the most comprehensive symbolic models of the protocol and is of independent interest. Its configurability makes it a strong foundation for automatically analyzing extensions and advanced properties of TLS 1.3. Our verification results for the secrecy and privacy of TLS 1.3 are also stronger than prior symbolic analyses. The next step is to obtain the missing privacy properties of TLS 1.3 (SIP,C-EXT) via the ECH extension.

4 TLS 1.3 WITH ENCRYPTED CLIENT HELLO

The Encrypted Client Hello (ECH) Extension [57] is an Internet Draft currently undergoing standardization at the IETF TLS Working Group. The extension has gone through 13 revisions (earlier versions were called ESNI for Encrypted SNI) and many of these revisions were motivated by attacks on prior designs.

The primary goal of the ECH extension is to hide the identity of the target TLS server (S_1 in Figure 1). Draft 0 of ESNI proposed a simple solution to this problem: we encrypt the SNI extension in the `ClientHello` with the public-key of the client-facing server F (say using HPKE [10]); F can then decrypt this extension before forwarding the `ClientHello` to the right server. The encryption uses the `ClientHello` nonce (cr) as associated data.

This simple design protects the server identity from passive network attackers but is vulnerable to active attacks. For example, a network attacker can simply replace the client's key share (g^x) with its own ($g^{x'}$) and it will then be able to decrypt the server's response to learn the server's certificate and hence the server's identity. This attack is depicted in Figure 3 and points to the crux of the design challenge: how does one encrypt the SNI such that it is *cryptographically bound* to the rest of the handshake.

Backward Binding. As the attack described above shows, an active network attacker can try to learn the SNI value by tampering with the ClientHello to observe the server's reaction. We may try to mitigate against this kind of attack by binding more of the contents of the ClientHello to the encrypted SNI. For example, adding the key share to the associated data of the encryption does prevent the attack of Figure 3, but still leaves open an attack based on PSKs.

Suppose the attacker has two pre-shared keys, one that it shares with S_1 and another that it shares with S_2. For example, these PSKs may be the output of ticket-based resumption handshakes that the attacker previously performed with S_1 and S_2. Then, when the client A sends a ClientHello with an encrypted SNI for S_1 or S_2 to F, the attacker can add its own PSK for (say) S_1 to this message with a correctly computed PSK binder. The server will decrypt the SNI value, retrieve the PSK, verify the binder, and will throw an error if the SNI value does not match the server name corresponding to the PSK. This tells the attacker which server the user was connecting to. This attack was discovered during the course of the analysis in this paper on draft 5 of ESNI, and is depicted in [16, Figure 6].

The current version of ECH (draft-13) takes a more systematic approach to backward binding by including the whole ClientHello within the encryption. Hence, the name Encrypted Client Hello.

Forward Binding. Binding the ECH just to the ClientHello is not enough to prevent active attacks. Recall the HelloRetryRequest flow of in normal TLS 1.3 from Figure 2, where a server can ask the client for a new ClientHello containing a new key share. Suppose a client tried to connect to S_1 or S_2 using ECH and the server responds with an HelloRetryRequest. Then the attacker can hijack this flow to insert its own encrypted ClientHello (say) for S_2 in response to the HelloRetryRequest. The server F will decrypt the attacker's (second) ClientHello and compare the server name with the client's (first) ClientHello and throw an error if they do not match, hence revealing the target server to the attacker. This attack is depicted in [16, Figure 8].

More generally, we would like to ensure that only the client who authored the first encrypted client hello is able to continue the handshake, which means we need forward binding between the ECH and the subsequent handshake. The ECH design achieves this in two ways. First, the nonce within the encrypted ClientHello (cr_i) serves as a secret unknown to the attacker that is mixed into the key schedule via the handshake transcript. Second, the HPKE encryptions for the first and second ClientHello must use the same HPKE context, which strongly binds them together and proves that both encryptions were computed by the same client.

ECH Protocol Design. The core idea behind the design of ECH is that we encrypt the entire ClientHello with the HPKE public key of the client-facing server. However, for practical deployments, the protocol must accommodate cases where the client-facing server rotates or forgets its private key. This has led to a design with two ClientHello messages, an inner one called ClientHelloInner and an outer one called ClientHelloOuter. ClientHelloInner contains the private parameters for the handshake with the *backend server* that a client wishes to keep secret from the attacker. ClientHelloOuter carries an HPKE encryption of ClientHelloInner as an extension, in addition to public parameters for completing the handshake with the *client-facing server* that are used when the client-facing server cannot decrypt ClientHelloInner. To prevent the attacker from tampering with the ClientHelloOuter, the full ClientHelloOuter (with the ECH extension replaced by a placeholder), is fed as associated data into the HPKE encryption of ClientHelloInner. The full changes introduced by ECH to the TLS 1.3 handshake are depicted (in red) in Figure 2.

Upon receiving a ClientHelloOuter, a server takes the following actions. If it does not support ECH or cannot decrypt the ClientHelloInner, it completes the handshake with ClientHello Outer. This branch is referred to as *rejecting* ECH. Otherwise, if the server successfully decrypts the extension, it forwards ClientHello Inner to the backend server (indicated by the SNI within Client HelloInner), which then completes the handshake. This branch is referred to as *accepting* ECH. The transcript used in the TLS key schedule varies depending on this branch. For ECH rejection the transcript includes ClientHelloOuter, which covers ClientHello Inner, whereas for ECH acceptance the transcript only includes ClientHelloInner.

Upon receiving the server's response, the client determines whether or not ECH was accepted and proceeds with the handshake accordingly. Servers indicate ECH acceptance in such a way that an attacker cannot distinguish between a successful handshake where ECH is accepted and a successful handshake where the client offered a fake ClientHello. Indicating acceptance is done with $accept_{sh}^{cr_i}$, which is computed as described in [16, Figure 3]. The output of this PRF is placed in the ServerHello.*random* field. Clients re-compute $accept_{sh}^{cr_i}$ and check it against ServerHello.*random*. When the values match, the client concludes acceptance, otherwise the client concludes rejection.

Alternatively, the server may respond with an HelloRetryRequest (asking for a new key share) along with an acceptance signal $accept_{sh}^{cr_i}$ included within an extension. The client would then send a new ClientHelloOuter with a new encrypted ClientHelloInner, but the encryption for the new ClientHelloInner uses the same HPKE context as the previous ClientHelloInner (using a sequence number 1 instead of 0). The server processes this new ClientHello Outer as before and responds with a ServerHello. In this Hello RetryRequest HelloRetryRequest protocol flow, the handshake transcript includes both the first and second ClientHello messages, so now we have four client hellos to reason about. The resulting transcript typically will either include the two ClientHello Inner values or the two ClientHelloOuters.

The protocol design also tries to hide whether a particular connection uses the ECH extension or not (a principle sometimes called "do not stick out"). Hence, a client that does not have the client-facing server's public key, will still send an ECH extension containing a random string whose contents are indistinguishable from an encrypted ClientHelloInner. This is referred to as a fake ClientHello or *grease*. We model grease in our work, but do not try to prove any extra privacy property for this feature.

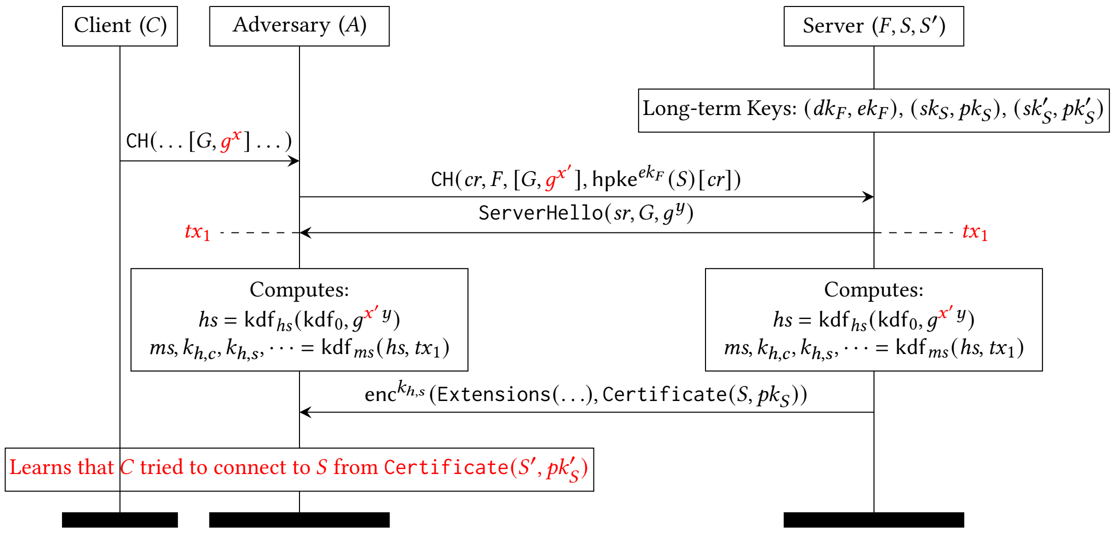
Client (C)
Adversary (A)
Server (F, S, S')
Long-term Keys: (dk_F, ek_F), (sk_S, pk_S), (sk'_S, pk'_S)
CH$(\dots [G, g^x] \dots)$
CH$(cr, F, [G, g^{x'}], \mathsf{hpke}^{ek_F}(S)[cr])$
ServerHello(sr, G, g^y)
tx_1
tx_1
Computes:
$hs = \mathsf{kdf}_{hs}(\mathsf{kdf}_0, g^{x'y})$
$ms, k_{h,c}, k_{h,s}, \dots = \mathsf{kdf}_{ms}(hs, tx_1)$
Computes:
$hs = \mathsf{kdf}_{hs}(\mathsf{kdf}_0, g^{x'y})$
$ms, k_{h,c}, k_{h,s}, \dots = \mathsf{kdf}_{ms}(hs, tx_1)$
$\mathsf{enc}^{k_{h,s}}$(Extensions$(\dots)$, Certificate(S, pk_S))
Learns that C tried to connect to S from Certificate(S', pk'_S)

Figure 3: Cut-and-paste attack (draft-ietf-tls-esni-00). $\mathsf{hpke}^{ek_F}(x)[y]$ **denotes fresh HPKE encryption under the public key ek_F of input x bound to value y.**

To summarize, ECH significantly changes the design of TLS 1.3 (as depicted in Figure 2) by modifying the handshake transcript and adding a new cryptographic construction (HPKE) that is bound in complex ways to the rest of the handshake. Considering these changes, and the attacks on prior versions of ECH, it becomes critical to study both the security and privacy of TLS 1.3 with ECH.

5 ANALYZING TLS 1.3+ECH USING PROVERIF

In this section, we describe our ProVerif model of TLS 1.3 with the ECH extension. We then show how we encode the security and privacy properties in this model. Finally, we describe the results of the analysis and its limitations. Our main goals for this analysis are to answer the following questions:

- Does TLS 1.3, with and without ECH, preserve the secrecy and authentication guarantees of TLS 1.3?
- Does TLS 1.3, with and without ECH, provide client identity privacy and unlinkability?
- Does TLS 1.3 with ECH provide server identity privacy and extension privacy?
- Can TLS 1.3 with ECH be downgraded to TLS 1.3?

We seek to answer these questions through an automated machine-checked symbolic analysis in ProVerif, for a configurable model that supports as many TLS 1.3 features as possible. Our full ProVerif models, with instructions for how to run the analysis, and a technical report describing our modeling strategies in detail, along with the full set of our results are provided in the supplementary material [1]. Here, we only describe the novel elements of our model and its analysis.

5.1 Modeling TLS 1.3 and ECH

Our ProVerif model consists of a series of cryptographic definitions for various constructions used in TLS 1.3 (see [16, Figure 3]) followed by process definitions for TLS 1.3 clients and servers that implement the protocol flows described in Figure 2 and [16, Figure 4]. Our model follows the overall structure of the prior ProVerif model [14] but is substantially rewritten to support configurability, extensions like ECH, and equivalence-based reasoning.

Modeling HPKE. In terms of cryptographic constructions, we define a new model in ProVerif of HPKE [10], an upcoming standard for hybrid public key encryption. Our model closely follows the standard and implements the sender-unauthenticated variant of HPKE used in ECH and in other protocols like MLS [9] and Oblivious DNS-over-HTTPS [46].

Modeling ECH. We first modify the TLS 1.3 client and server to include an HPKE-encrypted inner `ClientHello` as an extension within the outer `ClientHello` (see Figure 2). The inner `ClientHello` includes a nonce, a Diffie-Hellman key share, pre-shared key identifiers, an SNI extension (indicating the real target server S), and other (potentially secret) extensions. The outer `ClientHello` includes a different nonce and Diffie-Hellman key-share, a SNI extension (indicating the client-facing server F), and the encrypted inner `ClientHello`. We do not enable version negotiation or ciphersuite negotiation, although they are supported in our model and can be easily enabled (at significant cost to verification time).

We modify the treatment of `HelloRetryRequest` and `ServerHello` to include the ECH acceptance signal if the server supports and accepts ECH. When ECH is accepted, we use the inner `ClientHello` in the transcript. Otherwise, we use the outer `ClientHello`. With `HelloRetryRequest`, the definition of the transcript becomes more subtle since we now have four `ClientHello` messages to choose from. Depending on the protocol flow we may need to choose between the four combinations of outer-outer, inner-inner, inner-outer, and outer-inner, although the ECH specification admits only outer-outer or inner-inner as valid choices. Of course, it is crucial that the choice of transcript be the same at client and server and that an attacker cannot use this complexity to bypass the authentication guarantees of TLS 1.3 such as transcript agreement (AGR). Hence, even though ECH is a privacy-oriented extension, its secrecy and authentication guarantees need to be formally re-analyzed.

Configurable Features. A key novelty in our model is that our client and server processes are extensively configurable using two kinds of boolean flags: *global flags* enable or disable TLS features in all clients and processes, whereas *process flags* are used to enable features in specific client and server processes.

We define global flags for all the features listed in Table 1 in a configuration file that gets prefixed to each analyzed model. For example, the flag allow_HRR can be set to true to globally enable `HelloRetryRequest` in all clients and servers. We also define global flags that enable and disable ECH, PSK, tickets etc. and analysis-related flags that disable and enable the compromise of different kinds of keys used in the model.

Each client and server process is initialized with a set of arguments that can further configure their behavior. For example the definition of our TLS server process is as follows:

```
let server(use_ech, use_psk, cert_req:bool, s_dom:domain,
  g:group, h_alg:hash_alg, a_alg:aead_alg, s_extra:extraExt) = ...
```

This process defines a TLS 1.3 server configured with 8 arguments: use_ech specifies whether the server supports ECH, use_psk enables server support for PSK-DHE, cert_req says that the server should ask for a client certificate, s_dom is the server's domain name, g, h_alg, and a_alg define the server's preferred cryptographic algorithms, and s_extra contains additional (potentially secret) information the server can use in its `Extensions` message. In our model, each server process only supports one set of algorithms, chosen by the attacker from a given set of algorithms.

5.2 Encoding Security Goals

We define security goals for all the authentication, confidentiality, and privacy goals defined in Section 3.

Security Events. To precisely model our security goals, we annotate our client and server processes with events like ClientFinished and ServerFinished (indicating the completion of the handshake), and ServerPreFinished (indicating that the server has sent its `Finished` but not received the client `Finished`). We then add events like ClientSends, ServerReceives, ServerSends, and ClientReceives that indicate the sending and reception of application data. Each event is parameterized by arguments that include all the relevant connection parameters, including the identities of the client and server, their long-term keys, session identifiers, etc.

In addition to these protocol events, we define events that mark the compromise of long-term keys at the client and server. The attacker can ask for any HPKE private key, signature key, or pre-shared key stored in a database to be compromised, before leaking the key to the attacker, we trigger an event (e.g. CompromisedLtk).

Authentication, Integrity, and Downgrade Resilience. Each of our authentication goals (CAUTH, SAUTH, INT, AGR) are stated as ProVerif queries that express correspondences between events at the client and server. For example, the server authentication query (SAUTH) states that every time a client issues the event ClientReceives for a message m received from server S, it must either be the case that there was a matching event ServerSends issued by the server S for message m, or else either the private key of S (sk_S) (if the connection uses certificates) or the pre-shared key between C and S ($psk_{C,S}$) (if the connection uses PSKs) must be known to the adversary. The definition of CAUTH is similar.

In comparison to the prior ProVerif model of TLS 1.3 [14] our authentication goals for application data are stronger in three respects. First, we prove *injective* correspondence between events, hence proving that each client and server session maps to a *unique* event at a peer, whereas the prior analysis only proved non-injective correspondence. Second, we prove a stronger guarantee for ECH handshakes: even if the private-key or PSK of the server S is compromised, we obtain authentication as long as the HPKE private key of the client-facing server F remains secret. Third, we model and prove a novel stream integrity guarantee (INT) for application data: if a client or server receives a message m with sequence number n, then the peer must have sent the same message at the same sequence number, unless one of the compromise conditions hold. Hence, the client and server agree on the sequencing of messages in each direction, no matter how many messages they send or receive. In contrast, the prior model modeled authentication only for a single message at a time.

For handshake authentication, we state similar queries but in terms of ClientFinished, ServerFinished, and ServerPreFinished. We model key and transcript integrity (AGR) and key uniqueness (UNIQ) as correspondence queries over these events. We also define a limited form of downgrade resilience (DOWN): we ask that if an uncompromised TLS 1.3 client or server finishes a handshake with an uncompromised peer, and the connection does not use ECH, then either the client or the server did not support ECH. In other words, a network attacker cannot manipulate a handshake between an ECH client and an ECH server to disable ECH.

Secrecy and Indistinguishability. Key secrecy (SEC) and forward secrecy (FS) for 1RTT application data are written as queries over the attackers knowledge. For example, the forward secrecy query for 1RTT application data states that if the client issues an event ClientSends at time j before sending a message m to a server S, and if the attacker obtains the message m, written attacker(m), then it must be the case that the corresponding long-term key of S (either sk_S or $psk_{C,S}$) was compromised at some time $i < j$, and (if the connection uses ECH) then the HPKE private key of F must also have been compromised at some time $i' < j$. This formulation of forward secrecy is quite different and more precise than the query in the prior ProVerif model [14] since it now uses the recent timestamp feature of ProVerif [22] instead of coarse global phases.

Another novelty in our work is that we state and prove a stronger secrecy guarantee for the keys established by the handshake. All prior symbolic analyses of TLS 1.3, both in ProVerif [14] and in Tamarin [30, 31], only prove *syntactic* secrecy. This property means that an attacker cannot fully compute some secret, but even if the attacker computes all but one bit of the secret, this secrecy property would still hold. A stronger secrecy guarantee is to prove that the secret is *indistinguishable* from a random bitstring, but this kind of equivalence property is significantly harder to prove for symbolic tools, which is likely why it was omitted in prior work.

We show that the resumption PSK (*psk′*) generated by a TLS 1.3 handshake (with or without ECH) is indistinguishable from a fresh random PSK generated at the end of the handshake. In our model, this property is encoded as a *diff equivalence* in the client process, just before the secret is stored in the PSK database:

```
new psk_rand[]:preSharedKey;
let new_diff_psk = diff[new_psk,psk_rand] in
let new_diff_ticket_id = mk_idpsk(s_dom,h_alg,new_diff_psk) in
insert pre_shared_keys(c_dom,s_dom,h_alg,new_diff_ticket_id,
                       new_diff_psk,id_client,safe);
```

Here the PSK generated from the handshake is new_psk; we generate a fresh random PSK called psk_rand and define new_diff_psk as a left-or-right choice between these values, written diff[new_psk ,psk_rand]). In ProVerif syntax, this means that we would like to prove that the process where new_diff_psk is defined as new_psk is equivalent to the process where it is defined as psk_rand. In both cases, the PSK is stored in the pre_shared_keys table and hence may be used for resumption in future handshakes.

Recall that the attacker may compromise any long-term key, including PSKs. We only want to prove this equivalence in the case where the client and server of the current session is uncompromised. Otherwise, it trivially fails to hold. This restriction is encoded using flags like safe which indicate whether the keys in the current session are known to the adversary. This kind of flag is only used to specify security properties; it plays no part in the protocol execution.

Client Identity Privacy and Unlinkability. We model client identity privacy and unlinkability for TLS 1.3 (with and without ECH) also using diff-equivalences. We allow the attacker to choose the identity of the client before running the TLS 1.3 client process:

```
in(io,(clientA:name,clientB:name)); (* Attacker chooses client *)
let client_name = diff[clientA,clientB] in
client(id_client,use_ech,use_psk,client_name,server_name,algs)
```

This process says that running a client as clientA should be indistinguishable from running it as clientB, whether or not the client uses ECH, and also whether or not it uses certificates or PSKs for server and client authentication. Of course, this equivalence does not hold without further restrictions on the target scenario.

First, we assume that the server the client connects to has not been compromised (otherwise the attacker can impersonate the server and learn the client's identity). Second, we assume that either both clientA and clientB have a certificate they can authenticate with at the server, or neither of them do (otherwise the attacker can check if one of them fails to authenticate at the server). With these assumptions, we can prove client identity privacy. Third, we assume that either both clients have a valid PSK shared with the server or neither of them do (otherwise the attacker can detect the failure of a PSK handshake at one of the clients). Fourth, for TLS 1.3 handshakes without ECH, we need to assume that each PSK is used exactly once (otherwise the attacker can read the PSK identity from the client hello and use it to distinguish between clients.) With these four assumptions, we can prove both client identity privacy and unlinkability. That is, the client's connections cannot be linked and if it is authenticated then its name is kept secret.

The assumptions listed above are encoded using the *restrictions* syntax in ProVerif [22] that allow us to state assumptions about allowed traces. In effect, this allows us to define a very general model that allows all combinations of features and then restrict the cases to consider when analyzing a specific security goal.

Server Identity Privacy: Attacks and Proofs. The main goal of ECH is to protect the identity of the server. We encode it similarly to client privacy by using diff-equivalence over the server names:

```
let server_name = diff[backendA,backendB] in
client(id_client,use_ech,use_psk,client_name,server_name,algs)
```

Without using ECH (use_ech=**false**), this name is immediately revealed to the network attacker in the SNI extension and hence ProVerif shows that the equivalence is false. If we enable it and use an early variant of ESNI that only encrypts the server name, again ProVerif shows that the equivalence fails and produces the attack in Figure 3. Similarly, ProVerif can reconstruct the attacks of [16, Figures 6 and 8] as counter-examples to this equivalence property on models of earlier versions of ECH. Note that these attacks only appear when we model optional features of TLS 1.3 such as HRR, PSK, and SNI. So, we need to prove that the server identity privacy for ECH holds in a model that includes these features.

Server identity property only holds under the assumption that the HPKE private key of the client-facing server is uncompromised (otherwise the attacker can trivially decrypt the ECH extension to learn the backend name). Other than that, we allow all long-term keys at backend servers and clients to be compromised. When resumption is used, we additionally require that the client does not use a resumption PSK obtained from a session where the attacker impersonated backendA or backendB, e.g. by compromising their long-term keys (otherwise the attacker can detect the failure of a PSK handshake at the server it previously impersonated).

Extension Privacy. We model server extension privacy (S-EXT) for TLS 1.3, with and without ECH, as a diff-equivalence on the contents of the `Extensions` message, as chosen by the attacker. Client extension privacy (C-EXT) only makes sense for ECH handshakes and is also stated as a diff-equivalence on the contents of the extensions in the encrypted inner `ClientHello`. As expected, this property only holds under the assumption that the HPKE private key of the client-facing server is uncompromised.

5.3 ProVerif Analysis Results

Our full TLS 1.3 + ECH model totals 7600 lines. We set the resource limits of ProVerif to 48 hours of execution time and 100GB of memory and ran several analyses in parallel on a Linux server with a 64-core AMD processor clocked at 3.8Ghz with 515 GB RAM. We first ran ProVerif on our model with all supported features enabled for all the target properties (as summarized in Table 1). ProVerif succeeded in verifying all the reachability properties for TLS 1.3 without ECH, but ran out of time or memory in all other cases. We then ran ProVerif on various combinations of features for each property, resulting in 621 total analysis runs, from which we selected the best successful combinations. These selected results are depicted in Table 2, and the full results can be found in [1].

Verifying TLS 1.3. We first re-run all the classic TLS 1.3 secrecy and authentication reachability queries on our model of TLS 1.3 without ECH. We enable all optional features supported by our model, but

	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

	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

Table 2: Selected Verification Results for TLS 1.3 with ECH: reachability properties above, equivalence properties below. Each row represents a successful run of ProVerif on our model with a selection of features enabled. ECDHE handshakes with SNI are enabled in all runs (and so are omitted from the tables). For equivalence-based properties, 1-RTT and 0-RTT data are always disabled.

recall that we disable version and ciphersuite negotiation. This analysis completes in about 10 hours and validates our changes to the TLS 1.3 model of [14], while providing stronger authentication guarantees, as described above.

The analysis of equivalence-based properties typically takes more time and memory, and since these properties only speak about handshake elements, we disable application data (1RTT, 0RTT) when analyzing them. ProVerif proves key indistinguishability, server extension privacy, client identity privacy, unlinkability and for TLS 1.3 (without ECH, PHA, or application data) in 17 hours. We also proved CIP and UNL with HRR, PHA, CC and PSK-DHE but without tickets in 10 hours.

Reachability Properties for ECH. We then verify properties of TLS 1.3 with ECH. The added complexity of ECH makes the analysis much more expensive, requiring us to selectively disable certain features to obtain proofs for various properties. ProVerif is able to prove key secrecy (SEC) and key uniqueness (UNIQ) with 0RTT disabled. Application data has no impact on these two handshake properties, so disabling 0RTT does not weaken this result.

Secrecy of 0RTT data (SEC0) is then proved with client certificates (CC) disabled. Since certificates only appear after 0RTT data has already been sent, disabling it has no impact on this property.

Forward secrecy (FS) and stream integrity (INT) for 1RTT data are significantly more complex properties, and we can only prove them without HRR, PHA, and 0RTT. Enabling PHA would be interesting for these properties, since PHA is interleaved with 1RTT data, but ProVerif runs out of memory in this case.

Server authentication (SAUTH) and key and transcript agreement (AGR) are handshake properties and are proved with post-handshake (client) authentication and 0RTT disabled, neither of which have an impact on these properties.

Client authentication (CAUTH) is proved either with client certificates (CC) and PSKs (within the handshake) or for PHA, but not both at the same time. In practice, it is likely that TLS clients will either enable CC or PHA but not both. Still, ideally we would prove this property with all three client authentication modes (CC,PSK,PHA) enabled, but ProVerif runs out of memory.

Finally, ECH downgrade resilience is proved with certificate-based client authentication (CC,PHA) and 0RTT data disabled.

Equivalence Properties for ECH. For TLS 1.3 with ECH, PSK-DHE, tickets, and client certificates (but without HRR or PHA), key indistinguishability is proved in 21 hours. PHA has no impact on the handshake key schedule, but HRR does affect key computation. We separately prove indistinguishability for DHE handshakes with HRR, but without CC, PHA, and PSK-DHE. Put together, these results cover both certificate-based DHE handshakes and sequences of PSK-DHE handshakes with ticket-based resumption.

Server identity privacy is proved for handshakes with ECH and HRR, hence proving that ECH is not vulnerable to the attack of [16, Figure 8]. SIP is also proved for PSK handshakes handshakes with ticket-based resumption (TKT), hence proving the absence of the attack of [16, Figure 6]. Together these results cover all known privacy attacks on previous versions of ESNI and ECH.

Client identity privacy and unlinkability are proved for clients that use both PSKs and CC (but do not use HRR, PHA, or TKT.) These properties are additionally proved for handshakes that enable CC and PHA (but do not use PSK or TKT). These results cover both unauthenticated and authenticated clients that use client certificates, or PSKs, or post-handshake authentication.

Finally, server and client extension privacy are proved for handshakes with HRR, CC, and PSK-DHE but without TKT and PHA.

6 DISCUSSION

We have systematically analyzed TLS 1.3 with the ECH extension for a series of security and privacy properties in a variety of configurations. This analysis gives us more confidence in the design of ECH and our analysis has already influenced the design of the protocol. We discovered and presented attacks to the working group, participated in the protocol design for various versions, and are now in the process of presenting our findings to the group. Ours is the first formal security analysis for this privacy extension.

It is important to underline, however, that our verification results only hold in our model, and there is always a gap between a formal model and real-world deployments and concerns. Here we discuss some of these concerns and provide implementation and deployment guidelines for ECH implementors.

Network Configuration and Application Behavior. ECH does nothing to hide the IP address of the client-facing server. Thus, if client-facing servers orchestrate deployments such that each backend server is correlated or associated with a single IP address, then ECH offers little privacy benefits. Moreover, when considering the privacy of higher-level application behavior, such as a web page load which involves opening many TLS connections for subresources on a page, the *set* of IP addresses observed may leak information about any one subresource load. Patil and Borisov [53] analyze the effect of these "page load fingerprints" on the privacy of web browsing applications. Protecting against unique page load fingerprints requires more invasive changes either in the client network configuration, e.g., by sending all subresource connections over a VPN-like tunnel, or application-layer changes to alter what subresources are required for an application.

Do not stick out. One important criteria for ESNI and ECH to be effective is that use of it does not stick out. Indeed, early deployments of ESNI have seen some blocking in certain regions [24]. In order to ease ECH into deployment, it needs to be specifically designed so as to not obviously *stick out*. In other words, without prior information, an attacker should not be able to examine a TLS connection that *uses* the ECH extension and conclude that it indeed *negotiated* ECH without prior knowledge about the client-facing server or backend server. The ECH extension alone indicates *support* for the protocol, but does not indicate that ECH was actually negotiated without prior knowledge about the client-facing server configuration. Similarly, the size of the TLS handshake may leak whether ECH was negotiated. Our analysis does not attempt to model or prove this property, and we leave it for future work.

Traffic Analysis. More generally, most kinds of traffic analysis are out of scope of our model. Any attacks that rely on observing patterns of application messages, including their timing and lengths do not appear in our ProVerif analysis. Stating and proving privacy in the presence of traffic analysis is a hard but interesting problem, especially since it depends on network configuration and application behavior as described above.

ECH Implementation and Deployment Guidelines. Despite these limitations, TLS1.3+ECH implementations should still follow some implementation and deployment guidelines to obtain the maximal privacy guarantees from ECH.

First, our analysis shows that server privacy only holds when the client uses the same set of protocol parameters (ciphersuites, versions, groups) when connecting to both S_1 and S_2, otherwise the choice of algorithms may reveal which server it is talking to. This is already the case for most web browsers that offer a standard set of ciphersuites to all websites. However, in PSK resumption handshakes, the client may only offer the ciphersuite associated with the PSK. In this case, privacy only holds if the client holds PSKs with both S_1 and S_2 and both PSKs are associated with the same protocol parameters.

Second, the SNI extension in `ClientHello` can be of variable size; so the ECH specification recommends padding this value to avoid leaking information about the server. This should be implemented carefully to ensure that S_1 and S_2 have the same padded length.

Third, client-facing servers must ensure that all backend servers support the same set of cryptographic capabilities. This means that they all support the same set of cryptographic algorithms. In particular, if one backend server supports ticket-based resumption, they should all support this feature.

Fourth, the backend servers should pad their `Certificate` and `CertVerify` messages to ensure that any differences in certificate chains or handshake signatures do not leak via the length of the server's encrypted response. In general, the length of the server's encrypted response must be identical for all backend servers.

7 RELATED WORK AND CONCLUSION

We already discussed related work on TLS 1.3 in Section 3. Privacy aspects of secure channel, key exchange and related protocols, beyond TLS, have been studied in the literature. Lipp et al. [51] provided the first mechanised cryptographic proof of the WireGuard protocol including identity-hiding. Ramacher et al. [55] defined a privacy-preserving authenticated key exchange (PPAKE) protocol and prove that IKEv2 satisfies this definition. However, despite noting the importance of encrypted SNI in TLS as a step towards making it a PPAKE, they do not further analyze encrypted SNI designs for TLS 1.3. Zhao [60] defined an identity-concealed AKE (CAKE), and proposed a new cryptographic construction called *higncryption* to build it in the context of TLS 1.3. However, the CAKE definition is weaker than that of PPAKE since it does not allow one to analyze key indistinguishability separately from privacy. Dagdelen et al. [32] analyze OPACITY, though their analysis is limited in that each participant has one identity per protocol run. In contrast, TLS participants often have multiple identities. Fouque et al. [42] analyze *client* privacy in 3GPP, defined in terms of user identity confidentiality, service untraceability, and location untraceability. Among these notions, identity confidentiality is only relevant for the TLS handshake protocol.

More generally, privacy properties has been a major concern for several kinds of cryptographic protocols. However, the tool support for automated verification of privacy-type properties is far less mature than for authentication and confidentiality properties. Many recent works have improved support for equivalence-based properties in various tools [12, 22, 25, 26] and these tools have been used to analyse electronic voting protocols [27, 29], RFID protocols [43], 5G-AKA protocol [7], etc.

For most of these analyses, the protocol under study was relatively small or simplified for the tools to conclude. To our knowledge, our work is one of the most complex models for privacy properties ever proved for a large real-world protocol using symbolic analysis. Our results are at the limits of what ProVerif was able to prove for our model and we believe that they provide a good benchmark for future improvements to ProVerif. In particular, our analysis has shown that memory consumption is a bottleneck for large protocols and even more so for equivalence-based reasoning. We are currently experimenting with *hash consing* techniques to compactly represent messages in ProVerif which should allow us to re-enable features in our analysis. Moreover, this work led us to suggest improvements in the treatment of lemmas and restrictions in ProVerif, which have been accepted by the ProVerif authors for inclusion in the next version of the tool.

ACKNOWLEDGMENTS

This work has been partly supported by the ANR TECAP (decision number ANR-17-CE39-0004-03). K. Bhargavan received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 683032 - CIRCUS).

REFERENCES

[1] 2022. TLS ECH open-source repository. https://gitlab.inria.fr/chevalvi/echo_tls.

[2] Martín Abadi and Phillip Rogaway. 2000. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption).. In *IFIP TCS (Lecture Notes in Computer Science, Vol. 1872)*. Springer, 3–22.

[3] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, et al. 2015. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 5–17.

[4] Nadhem J. AlFardan and Kenneth G. Paterson. 2013. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *2013 IEEE Symposium on Security and Privacy (SP 2013)*. 526–540.

[5] Ghada Arfaoui, Xavier Bultel, Pierre-Alain Fouque, Adina Nedelcu, and Cristina Onete. 2019. The privacy of the TLS 1.3 protocol. *Proceedings on Privacy Enhancing Technologies* 2019, 4 (2019), 190–210.

[6] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. 2016. DROWN: Breaking TLS Using SSLv2. In *USENIX Security Symposium*. 689–706.

[7] David Baelde, Stéphanie Delaune, and Solène Moreau. 2020. A Method for Proving Unlinkability of Stateful Protocols. In *Proceedings of the 33rd IEEE Computer Security Foundations Symposium (CSF'20)*. IEEE Computer Society Press, Virtual conference, 169–183.

[8] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. 2021. SoK: Computer-Aided Cryptography. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 777–795.

[9] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. 2021. *The Messaging Layer Security (MLS) Protocol*. Internet-Draft draft-ietf-mls-protocol-12. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-mls-protocol-12 Work in Progress.

[10] Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A. Wood. 2021. *Hybrid Public Key Encryption*. Internet-Draft draft-irtf-cfrg-hpke-12. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hpke-12

[11] Richard Barnes, Bruce Schneier, Cullen Jennings, Ted Hardie, Brian Trammell, Christian Huitema, and Daniel Borkmann. 2015. Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement. RFC 7624. https://doi.org/10.17487/RFC7624

[12] David A. Basin, Jannik Dreier, and Ralf Sasse. 2015. Automated Symbolic Proofs of Observational Equivalence. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM, 1144–1155. https://doi.org/10.1145/2810103.2813662

[13] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. 2015. A Messy State of the Union: taming the Composite State Machines of TLS. In *IEEE Symposium on Security & Privacy (Oakland)*.

[14] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. 2017. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. 483–502.

[15] Karthikeyan Bhargavan, Christina Brzuska, Cédric Fournet, Matthew Green, Markulf Kohlweiss, and Santiago Zanella Béguelin. 2016. Downgrade Resilience in Key-Exchange Protocols. In *IEEE Symposium on Security and Privacy (Oakland)*. 506–525.

[16] Karthikeyan Bhargavan, Vincent Cheval, and Christopher Wood. 2022. *Handshake Privacy for TLS 1.3 - Technical report*. Technical report. https://gitlab.inria.fr/chevalvi/echo_tls/-/raw/master/Technical%20Report.pdf

[17] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jianyang Pan, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella-Béguelin, and Jean Zinzindohoué. 2017. Implementing and proving the TLS 1.3 record layer. In *SP 2017-38th IEEE Symposium on Security and Privacy*. 463–482.

[18] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. 2014. Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. In *IEEE Symposium on Security & Privacy (Oakland)*. 98–113.

[19] Karthikeyan Bhargavan and Gaëtan Leurent. 2016. On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 456–467.

[20] Karthikeyan Bhargavan and Gaetan Leurent. 2016. Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH. In *ISOC Network and Distributed System Security Symposium (NDSS)*.

[21] Bruno Blanchet. 2018. Composition Theorems for CryptoVerif and Application to TLS 1.3. In *IEEE Computer Security Foundations Symposium (CSF)*. 16–30.

[22] Bruno Blanchet, Vincent Cheval, and Véronique Cortier. 2022. ProVerif with lemmas, induction, fast subsumption, and much more. In *IEEE Symposium on Security and Privacy (S&P'22)*. IEEE Computer Society. To appear.

[23] Jacqueline Brendel, Marc Fischlin, and Felix Günther. 2019. Breakdown Resilience of Key Exchange Protocols: NewHope, TLS 1.3, and Hybrids. In *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11736)*, Kazue Sako, Steve A. Schneider, and Peter Y. A. Ryan (Eds.). Springer, 521–541.

[24] Zimo Chai, Amirhossein Ghafari, and Amir Houmansadr. 2019. On the importance of encrypted-SNI ({ESNI}) to censorship circumvention. In *9th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 19)*.

[25] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. 2018. DEEPSEC: Deciding Equivalence Properties in Security Protocols Theory and Practice. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 529–546. https://doi.org/10.1109/SP.2018.00033

[26] Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. 2017. SAT-Equiv: An Efficient Tool for Equivalence Properties. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*. IEEE Computer Society, 481–494. https://doi.org/10.1109/CSF.2017.15

[27] Véronique Cortier, David Galindo, and Mathieu Turuani. 2018. A Formal Analysis of the Neuchatel e-Voting Protocol. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*. IEEE, 430–442. https://doi.org/10.1109/EuroSP.2018.00037

[28] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. 2011. A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems. *J. Autom. Reason.* 46, 3–4 (apr 2011), 225–259.

[29] Véronique Cortier and Cyrille Wiedling. 2017. A formal analysis of the Norwegian E-voting protocol. *J. Comput. Secur.* 25, 1 (2017), 21–57. https://doi.org/10.3233/JCS-15777

[30] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. 2017. A comprehensive symbolic analysis of TLS 1.3. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1773–1788.

[31] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. 2016. Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication. In *IEEE Symposium on Security and Privacy (Oakland)*. 470–485.

[32] Özgür Dagdelen, Marc Fischlin, Tommaso Gagliardoni, Giorgia Azzurra Marson, Arno Mittelbach, and Cristina Onete. 2013. A cryptographic analysis of OPACITY. In *European Symposium on Research in Computer Security*. Springer, 345–362.

[33] David Peters. 2019. Breaking Bad – Are You Ready to Lawfully Intercept TLS 1.3? https://www.infosecurity-magazine.com/opinions/intercept-tls-13/.

[34] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella Béguelin, Karthikeyan Bhargavan, Jianyang Pan, and Jean Karim Zinzindohoue. 2017. Implementing and Proving the TLS 1.3 Record Layer. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. 463–482.

[35] Antoine Delignat-Lavaud, Cédric Fournet, Bryan Parno, Jonathan Protzenko, Tahina Ramananandro, Jay Bosamiya, Joseph Lallemand, Itsaka Rakotonirina, and Yi Zhou. 2021. A Security Model and Fully Verified Implementation for the IETF QUIC Record Layer. In *IEEE Symposium on Security and Privacy (Oakland)*. 1162–1178.

[36] D. Dolev and A. Yao. 2006. On the Security of Public Key Protocols. In *IEEE Trans. Inf. Theor.*, Vol. 29. 198–208.

[37] Jason A. Donenfeld. 2017. WireGuard: Next Generation Kernel Network Tunnel. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society.

[38] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. 2015. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. In *ACM Conference on Computer and Communications Security (CCS)*. 1197–1210.

[39] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. 2021. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol. *J. Cryptol.* 34, 4 (2021), 37.

[40] Nir Drucker and Shay Gueron. 2019. Selfie: reflections on TLS 1.3 with PSK. *IACR Cryptol. ePrint Arch.* 2019 (2019), 347. https://eprint.iacr.org/2019/347

[41] Marc Fischlin and Felix Günther. 2017. Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 60–75.

[42] Pierre-Alain Fouque, Cristina Onete, and Benjamin Richard. 2016. Achieving Better Privacy for the 3GPP AKA Protocol. *Proc. Priv. Enhancing Technol.* 2016, 4 (2016), 255–275.

[43] Lucca Hirschi, David Baelde, and Stéphanie Delaune. 2016. A method for verifying privacy-type properties: the unbounded case. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P'16)*, Michael Locasto, Vitaly Shmatikov,

and Úlfar Erlingsson (Eds.). IEEE Computer Society Press, San Jose, California, USA.

[44] Ian Levy. 2018. TLS 1.3: better for individuals - harder for enterprises. https://www.ncsc.gov.uk/blog-post/tls-13-better-individuals-harder-enterprises.

[45] Charlie Kaufman, Paul E. Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen. 2014. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296. https://doi.org/10.17487/RFC7296

[46] Eric Kinnear, Patrick McManus, Tommy Pauly, Tanya Verma, and Christopher A. Wood. 2022. *Oblivious DNS Over HTTPS.* Internet-Draft draft-pauly-dprive-oblivious-doh-09. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-pauly-dprive-oblivious-doh-09 Work in Progress.

[47] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. 2015. (De-)Constructing TLS 1.3. In *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9462)*, Alex Biryukov and Vipul Goyal (Eds.). 85–102.

[48] Hugo Krawczyk. 2003. SIGMA: The 'SIGn-and-MAc'approach to authenticated Diffie-Hellman and its use in the IKE protocols. In *Annual International Cryptology Conference.* Springer, 400–425.

[49] Hugo Krawczyk and Hoeteck Wee. 2016. The OPTLS Protocol and TLS 1.3. In *IEEE European Symposium on Security & Privacy (Euro S&P).* Cryptology ePrint Archive, Report 2015/978.

[50] X. Li, J. Xu, Z. Zhang, D. Feng, and H. Hu. 2016. Multiple Handshakes Security of TLS 1.3 Candidates. In *IEEE Symposium on Security and Privacy (Oakland).* 486–505.

[51] Benjamin Lipp, Bruno Blanchet, and Karthikeyan Bhargavan. 2019. A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol. In *IEEE European Symposium on Security and Privacy (EuroS&P).* IEEE, 231–246.

[52] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. 2014. This POODLE bites: exploiting the SSL 3.0 fallback. https://www.openssl.org/~bodo/ssl-poodle.pdf.

[53] Simran Patil and Nikita Borisov. 2019. What can you learn from an IP?. In *Proceedings of the Applied Networking Research Workshop.* 45–51.

[54] Trevor Perrin. 2018. The Noise Protocol Framework. http://noiseprotocol.org/noise.html.

[55] Sebastian Ramacher, Daniel Slamanig, and Andreas Weninger. 2021. Privacy-Preserving Authenticated Key Exchange: Stronger Privacy and Generic Constructions. In *European Symposium on Research in Computer Security.* Springer, 676–696.

[56] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446

[57] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. 2021. *TLS Encrypted Client Hello.* Internet-Draft draft-ietf-tls-esni-13. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-10 Work in Progress.

[58] The SSL Store. 2018. TLS 1.3: Banking Industry Working to Undermine Encryption. https://medium.com/@thesslstore/tls-1-3-banking-industry-working-to-undermine-encryption-752838cf828c.

[59] Mathy Vanhoef and Frank Piessens. 2015. All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS. In *USENIX Security Symposium.* 97–112.

[60] Yunlei Zhao. 2016. Identity-concealed authenticated encryption and key exchange. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.* 1464–1479.