
EuroSP'21 | reprosec.org

Bhargavan, Bichhawat, Do, Hosseyni, Küsters, Schmitz, Würtele
Cryptographic Protocols are Everywhere: **Golden Era of Crypto**

- Ubiquitous HTTPS: TLS 1.3, QUIC, ACME/Let’s Encrypt, ...
- Secure Messaging: Signal, MLS, ...
- Single-Sign On: OAuth, OIDC, SAML, ...
- Wireless: Wifi/WPA, 4G, 5G, Zigbee, ...
- Payment: EMV, W3C Web Payments, ...
- Post-Quantum Crypto: NIST KEMs, Signature, ...
- Lightweight Crypto: IETF LAKE, NIST LWC
Cryptographic Protocols are Everywhere: Golden Era of Crypto

- Ubiquitous HTTPS: TLS 1.3, QUIC, ACME/Let’s Encrypt, …
- Secure Messaging: Signal, MLS, …
- Single-Sign On: OAuth, OIDC, SAML, …
- Wireless: Wifi/WPA, 4G, 5G, Zigbee, …
- Payment: EMV, W3C Web Payments, …
- Post-Quantum Crypto: NIST KEMs, Signature, …
- Lightweight Crypto: IETF LAKE, NIST LWC
Cryptographic Protocols are Everywhere: **Golden Era of Crypto**

- Ubiquitous HTTPS: TLS 1.3, QUIC, ACME/Let's Encrypt, …
- Secure Messaging: Signal, MLS, …
- Single-Sign On: OAuth, OIDC, SAML, …
- Wireless: Wifi/WPA, 4G, 5G, Zigbee, …
- Payment: EMV, W3C Web Payments, …
- Post-Quantum Crypto: NIST KEMs, Signature, …
- Lightweight Crypto: IETF LAKE, NIST LWC
Cryptographic Protocols are Everywhere: Golden Era of Crypto

- Ubiquitous HTTPS: TLS 1.3, QUIC, ACME/Let’s Encrypt, …
- Secure Messaging: Signal, MLS, …
- Single-Sign On: OAuth, OIDC, SAML, …
- Wireless: Wifi/WPA, 4G, 5G, Zigbee, …
- Payment: EMV, W3C Web Payments, …
- Post-Quantum Crypto: NIST KEMs, Signature, …
- Lightweight Crypto: IETF LAKE, NIST LWC
Cryptographic Protocols are Everywhere: Golden Era of Crypto

- Ubiquitous HTTPS: TLS 1.3, QUIC, ACME/Let's Encrypt, …
- Secure Messaging: Signal, MLS, …
- Single-Sign On: OAuth, OIDC, SAML, …
- Wireless: Wifi/WPA, 4G, 5G, Zigbee, …
- Payment: EMV, W3C Web Payments, …
- Post-Quantum Crypto: NIST KEMs, Signature, …
- Lightweight Crypto: IETF LAKE, NIST LWC
Cryptographic Protocols are Everywhere: **Golden Era of Crypto**
Signal Messaging Protocol

- Asynchronous continuous key exchange

\[ G = \langle g \rangle \]

\[ g^x \]

\[ x \text{ random} \]

\[ (g^y)^x = g^{xy} \]

\[ g^y \]

\[ y \text{ random} \]

\[ (g^x)^y = g^{xy} \]
Signal Messaging Protocol

- Asynchronous continuous key exchange
Signal Messaging Protocol

- Asynchronous continuous key exchange
- Multiple subprotocols
  - X3DH (initial key exchange)
  - DH Ratchet (post-compromise security)
  - Hash Ratchet (forward security)
  - Authenticated Encryption (message security)
Signal Messaging Protocol

- Asynchronous continuous key exchange
- Multiple subprotocols
  - X3DH (initial key exchange)
  - DH Ratchet (post-compromise security)
  - Hash Ratchet (forward security)
  - Authenticated Encryption (message security)
- Inherently recursive
  - Security of each message depends on a chain of derived keys
Signal Messaging Protocol

- Asynchronous continuous key exchange
- Multiple subprotocols
  - X3DH (initial key exchange)
  - DH Ratchet (post-compromise security)
  - Hash Ratchet (forward security)
  - Authenticated Encryption (message security)
- Inherently recursive
  - Security of each message depends on a chain of derived keys
- Can we mechanically verify that the protocol is secure?
Formalizing Signal

**Initiator I**

Prior Knowledge:
\( (i.g') \)

Initiate\((i.g', g'(y'), y'(y''))) \rightarrow (r_{b_0})
\( \text{generic } (c, g') \)
\( db_0 = \text{Hash}(y') \)
\( r_{b_0} = \text{HashKey}(db_0, 0x0000, "WhisperRatcheting") \)

SendRatchet\((r_{b_0}, y') \rightarrow (r_{b_1}, c_0, t_0)\):
\( \text{generate } (x_0, y'') \)
\( r_{b_1} = \text{HashKey}(y'', r_{b_0}, "WhisperRatcheting") \)

Encrypt\((y', g', y'', c_0, t_0) \rightarrow (c_{k_0}, c_0, t_0)\):
\( c_{k_0} = \text{HashAC}(c_0, 0x01) \)
\( k_{b_0} = \text{HashAC}(c_{k_0}, 0x00) \)
\( c_0 = \text{HashKey}(y'', 0x0, \text{AES-CBC}(\text{HashAC}(c_0, 0x00)) \)
\( t_0 = \text{HashAC}(\text{HashAC}(c_0, 0x00), 0x0339) \)

**Responder R**

Prior Knowledge:
\( (r_{b_0}, g', y', y'') \)

Respond\((r_{b_0}, g', y') \rightarrow (r_{b_2})\):
\( \text{recompute } db_0, r_{b_0} \text{ similarly to } I \)

ReceiveRatchet\((r_{b_0}, y'', c_0, t_0) \rightarrow (r_{b_1}, c_{k_0})\):
\( \text{recompute } r_{b_1}, c_{k_0} \text{ similarly to } I \)

Decrypt\((c_{k_0}, c_0, t_0) \rightarrow (m_0, c_{k_0})\):
\( \text{recompute } c_{k_0}, t_0 \text{ similarly to } II \)
\( \text{verify MAC and decrypt } (c_{k_0}, t_0) \) to get \( m_0 \)

SendRatchet\((r_{b_1}, y'') \rightarrow (r_{b_2}, c_{k_0})\)

Encrypt\((y', g', y''', c_{k_0}, m_0) \rightarrow (c_{k_1}, c_1, t_1)\)

Session State:
\( \text{root_key: } r_{b_0}, \text{send_chain: } (c_{k_0}, t_0), \text{recv_chain: } (c_{k_1}, t_1) \)

Session State:
\( \text{root_key: } r_{b_2}, \text{send_chain: } (c_{k_1}, t_1), \text{recv_chain: } (c_{k_1}, y''') \)
Formalizing Signal

- **Existing Analyses**
  - Using ProVerif and CryptoVerif
  - Model X3DH, Double Ratchet
  - Few hundred lines written in applied pi calculus
Formalizing Signal

- Existing Analyses
  - Using ProVerif and CryptoVerif
  - Model X3DH, Double Ratchet
  - Few hundred lines written in applied pi calculus

- One major limitation of existing analyses: Proofs for only 3 message rounds due to recursion
Analysis of Security Protocols: Tools

Computational Tools: CryptoVerif, EasyCrypt, ...
- Focus on cryptographic core
- Messages are bitstrings
- Probabilistic

Symbolic Tools: ProVerif, Tamarin, RCF, ...
- Abstract cryptography
- Messages are formal terms
Analysis of Security Protocols: Tools

Computational Tools: CryptoVerif, EasyCrypt, ...

- Focus on cryptographic core
- Messages are bitstrings
- Probabilistic

Symbolic Tools: ProVerif, Tamarin, RCF, ...

- Abstract cryptography
- Messages are formal terms
Existing Symbolic Approaches and DY*

DY-style tools:
- Tamarin
- ProVerif

Dependent Types:
Existing Symbolic Approaches and DY*

DY-style tools:
Tamarin, ProVerif, ...

Dependent Types:
RCF, F7, ...
Existing Symbolic Approaches and DY*

DY-style tools: Tamarin, ProVerif, ...

Dependent Types: RCF, F7, ...

Focus on protocol core

- Abstract models
- Bounded data structures
- No modularity
- Limited inductive reasoning
- Interoperability

✔ Automated analysis
   (potentially some user interaction)

✔ Global trace & properties

✔ Equational theories
Existing Symbolic Approaches and DY*

DY-style tools:
- Tamarin, ProVerif, ...

focus on protocol core

- abstract models
- bounded data structures
- no modularity
- limited inductive reasoning
- interoperability

✓ automated analysis (potentially some user interaction)
✓ global trace & properties
✓ equational theories

✓ modular proofs
✓ implementation level analysis
✓ unbounded structures
✓ inductive reasoning
✓ executable models
✓ interoperability

Dependent Types:
- RCF, F7, ...

focus on implementation aspects

✗ missing global view
✗ limited expressivity w.r.t. security prop.
✗ limited support for mutable state
✗ less automation
✗ no equational theories (e.g., DH)
## Existing Symbolic Approaches and DY*

<table>
<thead>
<tr>
<th>Existing Symbolic Approaches</th>
<th>DY*</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗ abstract models</td>
<td>✔</td>
</tr>
<tr>
<td>✗ bounded data structures</td>
<td>✔</td>
</tr>
<tr>
<td>✗ no modularity</td>
<td>✔</td>
</tr>
<tr>
<td>✗ limited inductive reasoning</td>
<td>✔</td>
</tr>
<tr>
<td>✗ interoperability</td>
<td>✔</td>
</tr>
<tr>
<td>✔ automated analysis</td>
<td>✔</td>
</tr>
<tr>
<td>(potentially some user interaction)</td>
<td></td>
</tr>
<tr>
<td>✔ global trace &amp; properties</td>
<td>✔</td>
</tr>
<tr>
<td>✔ equational theories</td>
<td>✔</td>
</tr>
<tr>
<td>✔ modular proofs</td>
<td>✔</td>
</tr>
<tr>
<td>✔ implementation level analysis</td>
<td>✔</td>
</tr>
<tr>
<td>✔ unbounded structures</td>
<td>✔</td>
</tr>
<tr>
<td>✔ inductive reasoning</td>
<td>✔</td>
</tr>
<tr>
<td>✔ executable models</td>
<td>✔</td>
</tr>
<tr>
<td>✔ interoperability</td>
<td>✔</td>
</tr>
</tbody>
</table>

**DY-style tools:** Tamarin, ProVerif, ...

**Dependent Types:** RCF, F7, ...

**DY-style tools:** Tamarin, ProVerif, ...

**focus on protocol core**

**focus on implementation aspects**

**missing global view**

**limited expressivity w.r.t. security prop.**

**limited support for mutable state**

**less automation**

**no equational theories (e.g., DH)**
What is F*?

• Functional programming language aimed at program verification
  - Can be used to precisely express strong (security) properties

• Developed and actively supported by Microsoft Research, INRIA, and others

• Already used for computational protocol analysis (for example, parts of TLS 1.3)

• Rich, versatile type system
  - Dependent and refinement types
  - Backed by SMT-Solver Z3
  - Pre/post conditions
  - Allow modeling unbounded and recursive data structures
Dolev-Yao* (DY*): Architecture

Append-only log that captures relevant interaction with the framework.
Dolev-Yao* (DY*): Architecture

Append-only log that captures relevant interaction with the framework.
Dolev-Yao* (DY*): Architecture

- Crypto
- Network Communication
- Application State

Append-only log that captures relevant interaction with the framework.
Dolev-Yao* (DY*): Architecture

Append-only log that captures relevant interaction with the framework.
Dolev-Yao* (DY*): Architecture

Append-only log that captures relevant interaction with the framework.

Runtime Model

Crypto

Network Communication

Application State

Global Trace

Protocol Implementation
Dolev-Yao* (DY*): Architecture

Runtime Model

Crypto

Network Communication

Application State

Labeling Layer

Generic Proofs

Protocol Implementation

Append-only log that captures relevant interaction with the framework.

Global Trace
Dolev-Yao* (DY*): Architecture

- Runtime Model
- Labeling Layer
- Crypto
- Network Communication
- Application State
- Security Properties
- Append-only log that captures relevant interaction with the framework.
- Generic Proofs
- Protocol Implementation

Global Trace
Dolev-Yao* (DY*): Architecture

Runtime Model

Crypto

Network Communication

Application State

Global Trace

Append-only log that captures relevant interaction with the framework.

Security Properties

Generic (e.g., attackers) + Application-specific invariants

Labeling Layer

Generic Proofs

INvariants

Protocol Implementation

Global Trace

Append-only log that captures relevant interaction with the framework.
Dolev-Yao* (DY*): Architecture

- **Runtime Model**
  - Crypto
  - Network
  - Communication
  - Application
  - State

- **Labeling Layer**
  - Generic Proofs

- **Security Properties**
  - Append-only log that captures relevant interaction with the framework.

- **Generic (e.g., attackers) + Application-specific invariants**

- **Global Trace**

- **INVAriANTS**
Dolev-Yao* (DY*): Architecture

Append-only log that captures relevant interaction with the framework.

Runtime Model

Labeling Layer

Crypto

Network

Communication

Application State

F* verifies that application preserves invariants

Generic Proofs

Protocol Implementation

INVENTARS

Global Trace

Security Properties

Generic (e.g., attackers) + Application-specific invariants

INVENTARS

Append-only log that captures relevant interaction with the framework.

F* verifies that application preserves invariants
Case Studies

• Signal Messaging Protocol
Signal Messaging Protocol

- First mechanized proof accounting for:
  - Forward Secrecy
  - Post-compromise Security
  - Unbounded number of protocol rounds at the same time

- First type-based formulation and proof of post-compromise security for any protocol

- First analysis of Signal based on dependent types
Case Studies

• **Signal Messaging Protocol**
  - Unbounded number of rounds (ratcheting)
  - Forward Secrecy & Post Compromise Security

• **Needham-Schroeder(-Lowe), ISO-DH, and ISO-KEM**
Conclusion & Future Work

- **Golden era** of cryptographic protocols

- We recently proposed DY*, a **new mechanized symbolic verification framework** for protocols and their code
  - Overcomes many limitations of existing tools
  - Precise reasoning on global properties
  - Account for low-level protocol details
  - Protocol models can even be interoperable
Conclusion & Future Work

- **Golden era** of cryptographic protocols
- We recently proposed DY*, a new mechanized symbolic verification framework for protocols and their code
  - Overcomes many limitations of existing tools
  - Precise reasoning on global properties
  - Account for low-level protocol details
  - Protocol models can even be interoperable
- Lots of interesting work to be done!
  - Equivalence properties
  - Computational analysis
  - WIM*: mechanize the Web Infrastructure Model

Conclusion & Future Work

- **Golden era** of cryptographic protocols
- We recently proposed DY*, a **new mechanized symbolic verification framework** for protocols and their code
  - Overcomes many limitations of existing tools
  - Precise reasoning on global properties
  - Account for low-level protocol details
  - Protocol models can even be interoperable

- **Lots of interesting work to be done!**
  - Equivalence properties
  - Computational analysis
  - WIM*: mechanize the **Web Infrastructure Model**

Find more information on: [reprosec.org](http://reprosec.org)

Thank you!