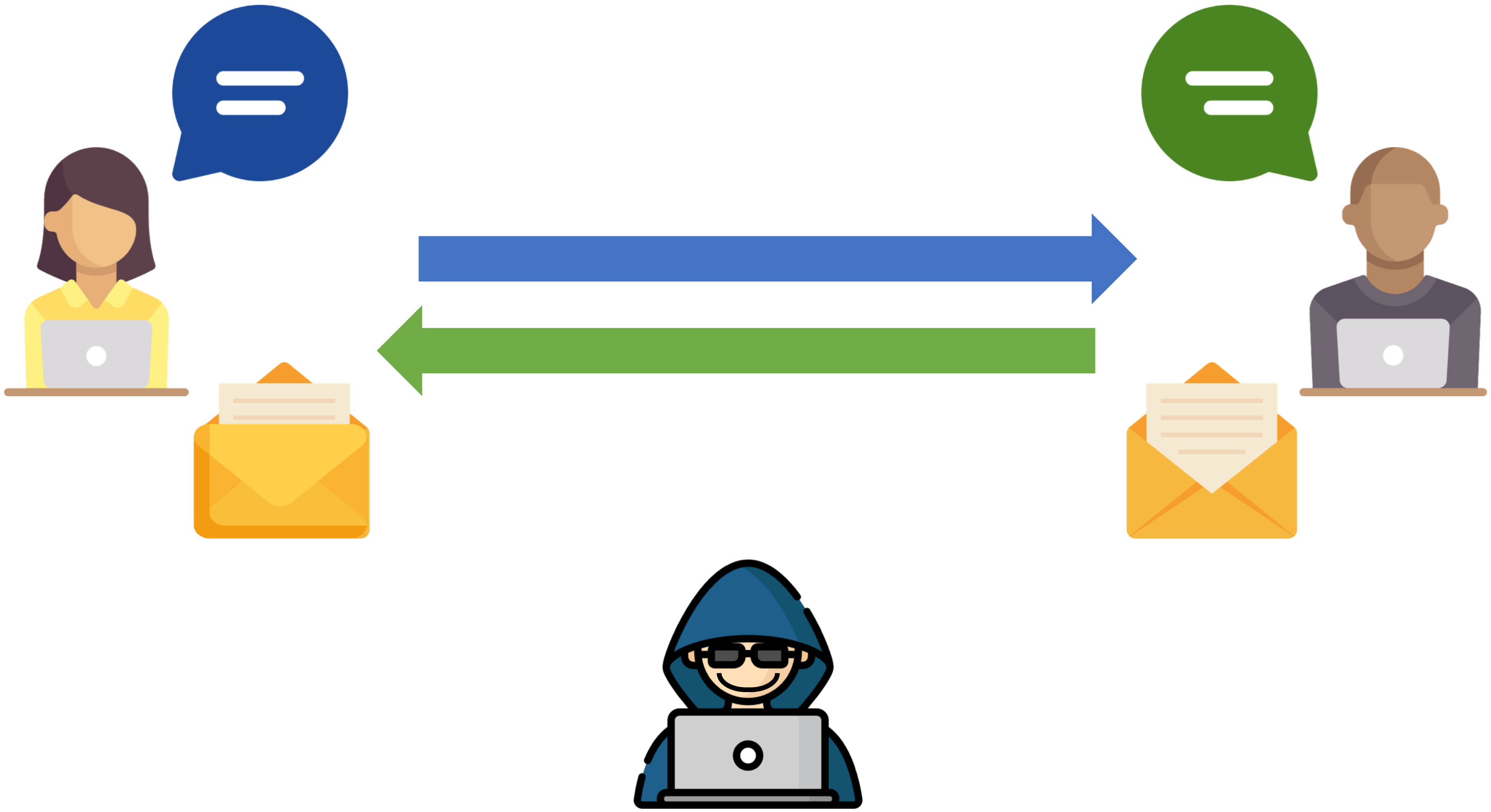
Are you messaging securely?

Abhishek Bichhawat

Secure Messaging

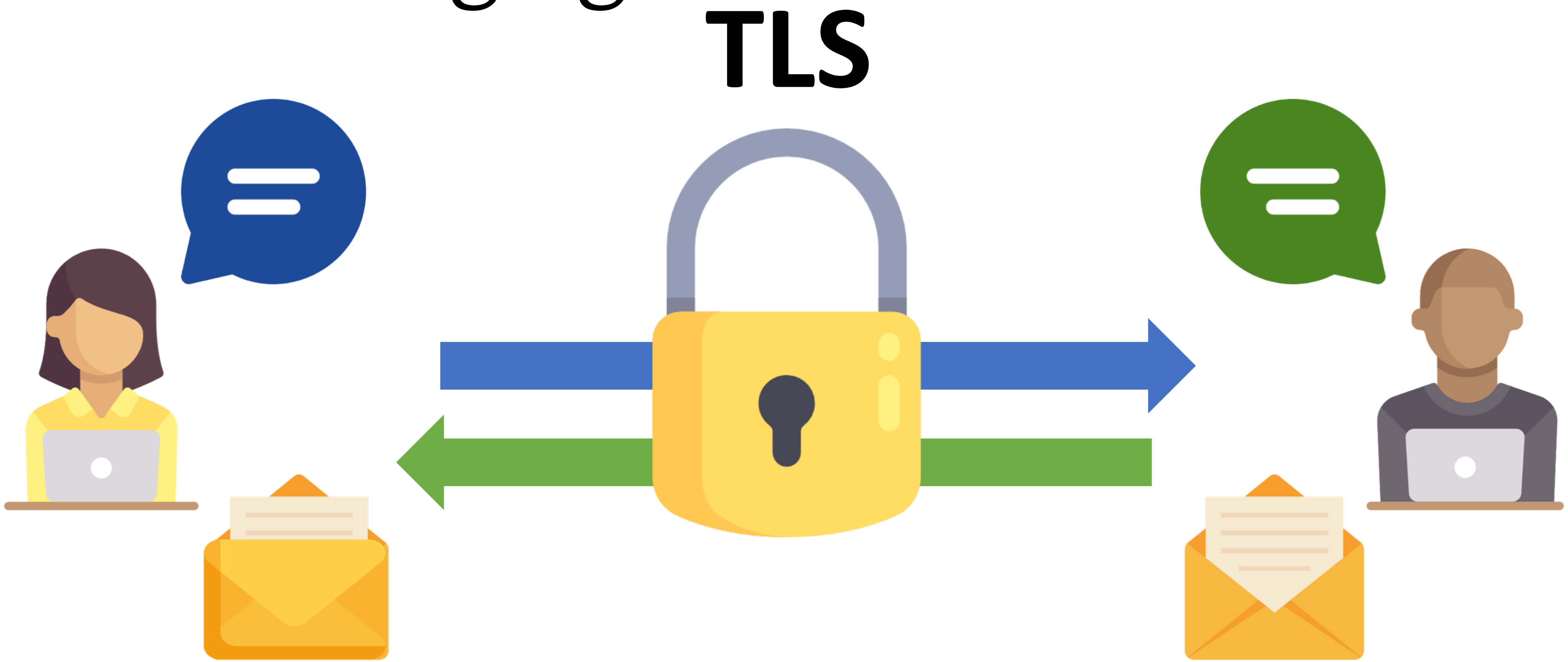


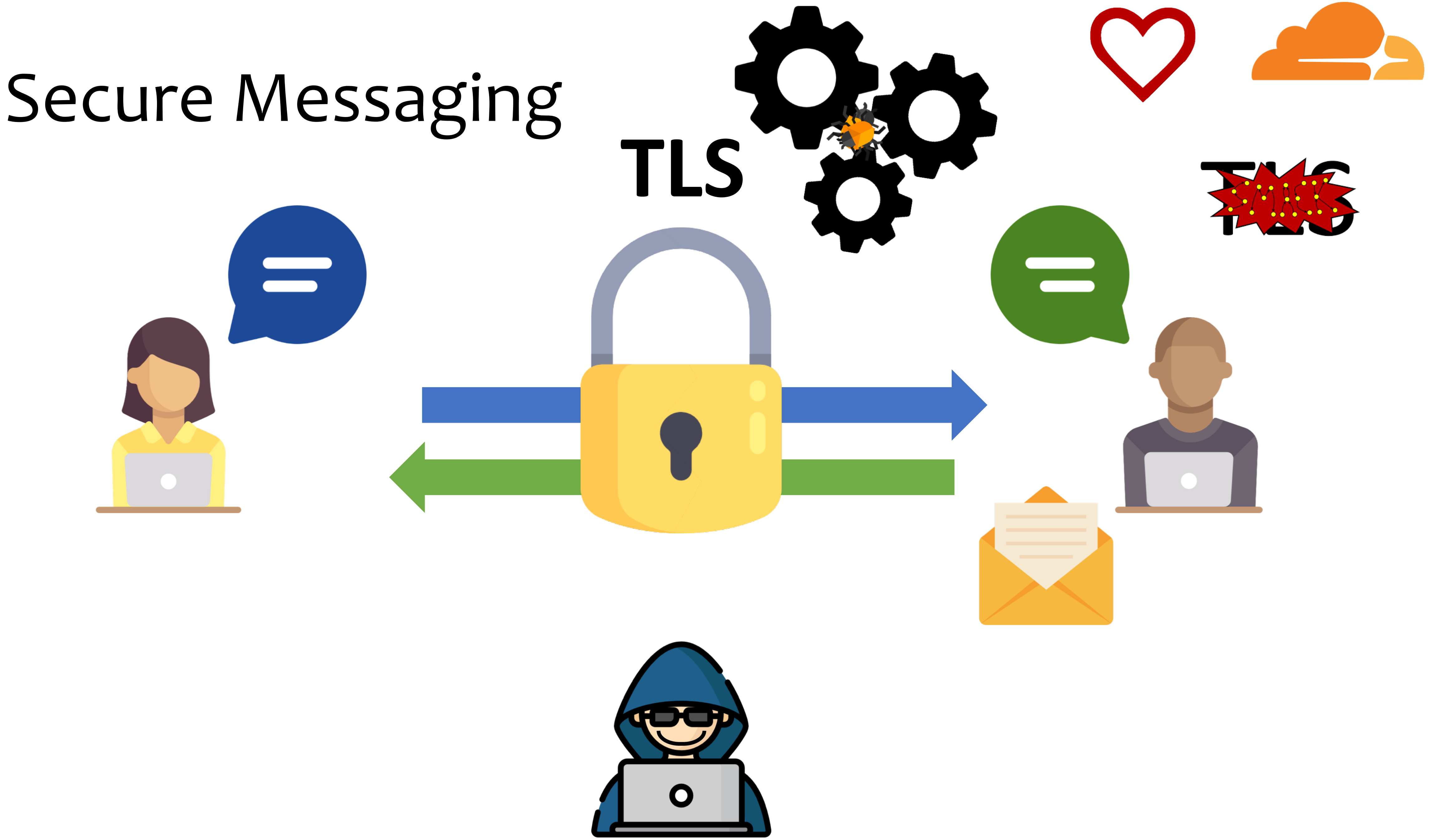
Secure Messaging

Cryptographic Protocols are Everywhere

- 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, ...

Secure Messaging




Secure Messaging
TLS

(In)Secure Messaging

- Lowe's attack and fix of Needham-Schroeder public key protocol ('95)

ELSEVIER

Information Processing Letters

Volume 56, Issue 3, 10 November 1995, Pages 131-133

An attack on the Needham-Schroeder public-key authentication protocol

Gavin Lowe

(In)Secure Messaging

Verifying Protocols

- Formal analysis of cryptographic protocols
  - Lowe showed that his fix was sufficient using a symbolic tool

Computational Tools:	**Symbolic Tools:**
CryptoVerif, EasyCrypt	*ProVerif, Tamarin*
Precise probabilistic assumptions of primitives	Abstract notions of crypto primitives
More precise; more effort	Scale better
Infeasible for large protocols	Less precise details about the primitives used

Tools for Verifying Protocols

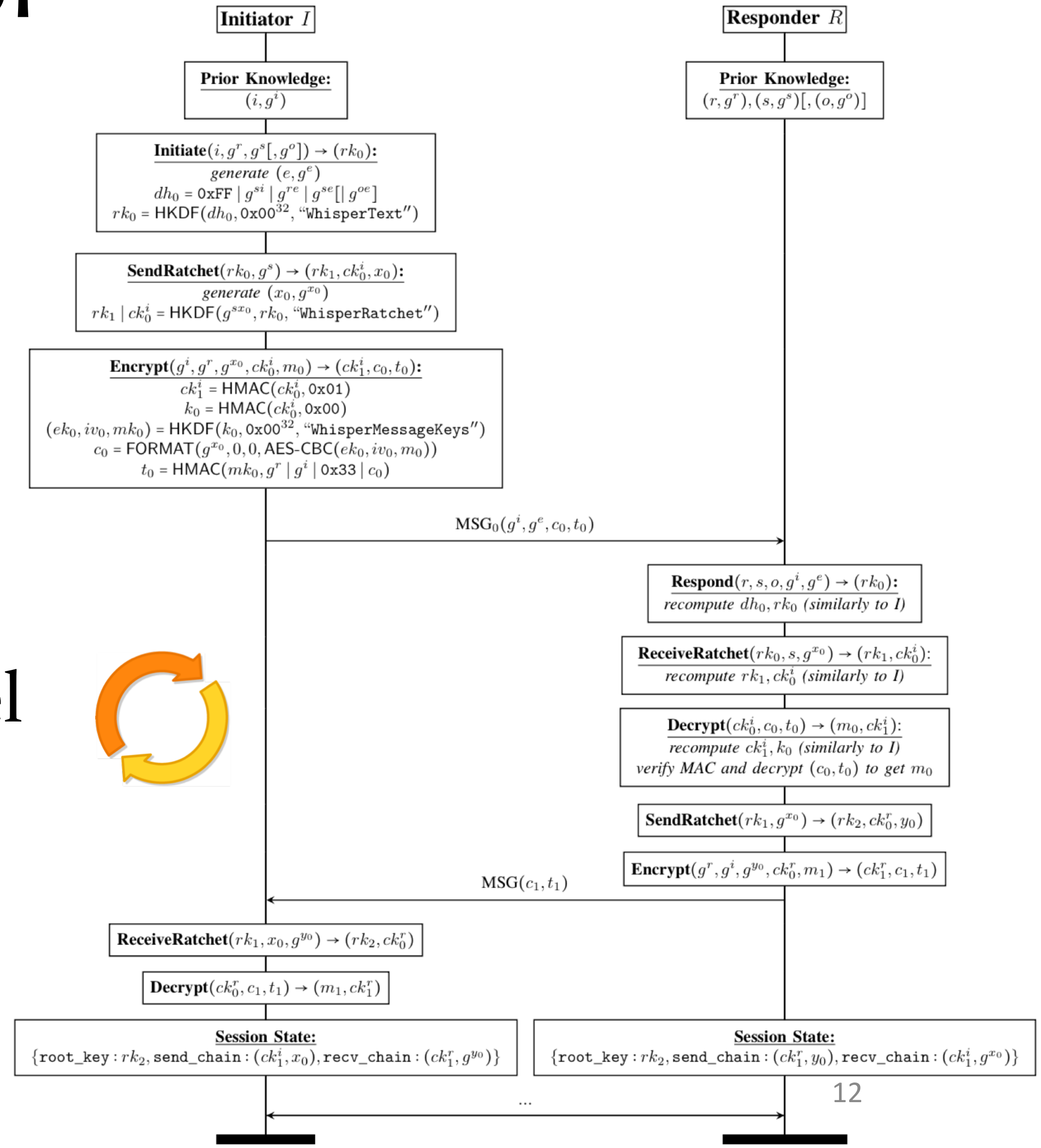
- Automated symbolic protocol analysis
  - Analyze all possible execution traces
  - Do not scale well for complex protocols
    - Perform whole protocol analysis
    - Cannot break the analysis into smaller (re-usable) modules.
  - Protocols with unbounded loops and recursive data structures are challenging to model in these tools
  - Models are too abstract and often leave out important implementation details

Signal Messaging Protocol

- Asynchronous continuous key exchange protocol
- 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

- Existing Analyses
  - using ProVerif and CryptoVerif
  - Model X3DH, Double Ratchet
  - Few hundred lines written in applied pi calculus
- ProVerif uses symbolic abstraction
- CryptoVerif uses computational model
- One major limitation
  - Proofs for only 3 message rounds due to recursion

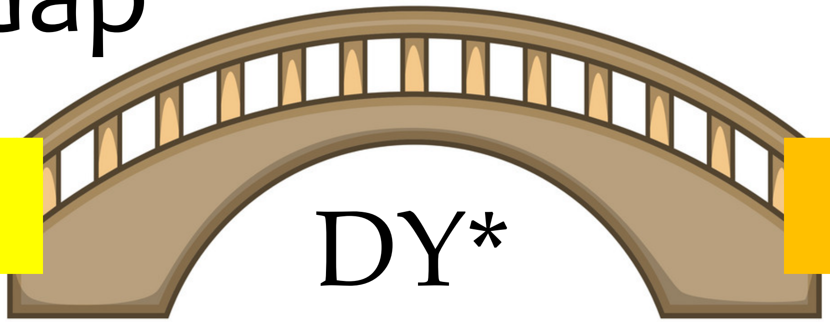


Tools for Verifying Protocols

- Dependent type systems based analysis
  - E.g. RCF, F7 etc.
  - Provide modular proofs
  - Implementation level analysis with unbounded structures
  - Provide executable models with interoperability
  - Less automation
  - No equational theories (do not model Diffie-Hellman/XOR)

Bridging the Gap

DY-style tools:
Tamarin, ProVerif, …



Dependent Types:
RCF, F7, …

focus on protocol core

focus on implementation aspects

abstract models
bounded data structures
no modularity
limited inductive reasoning
limited executability

(mostly) automated analysis
global trace & properties
equational theories

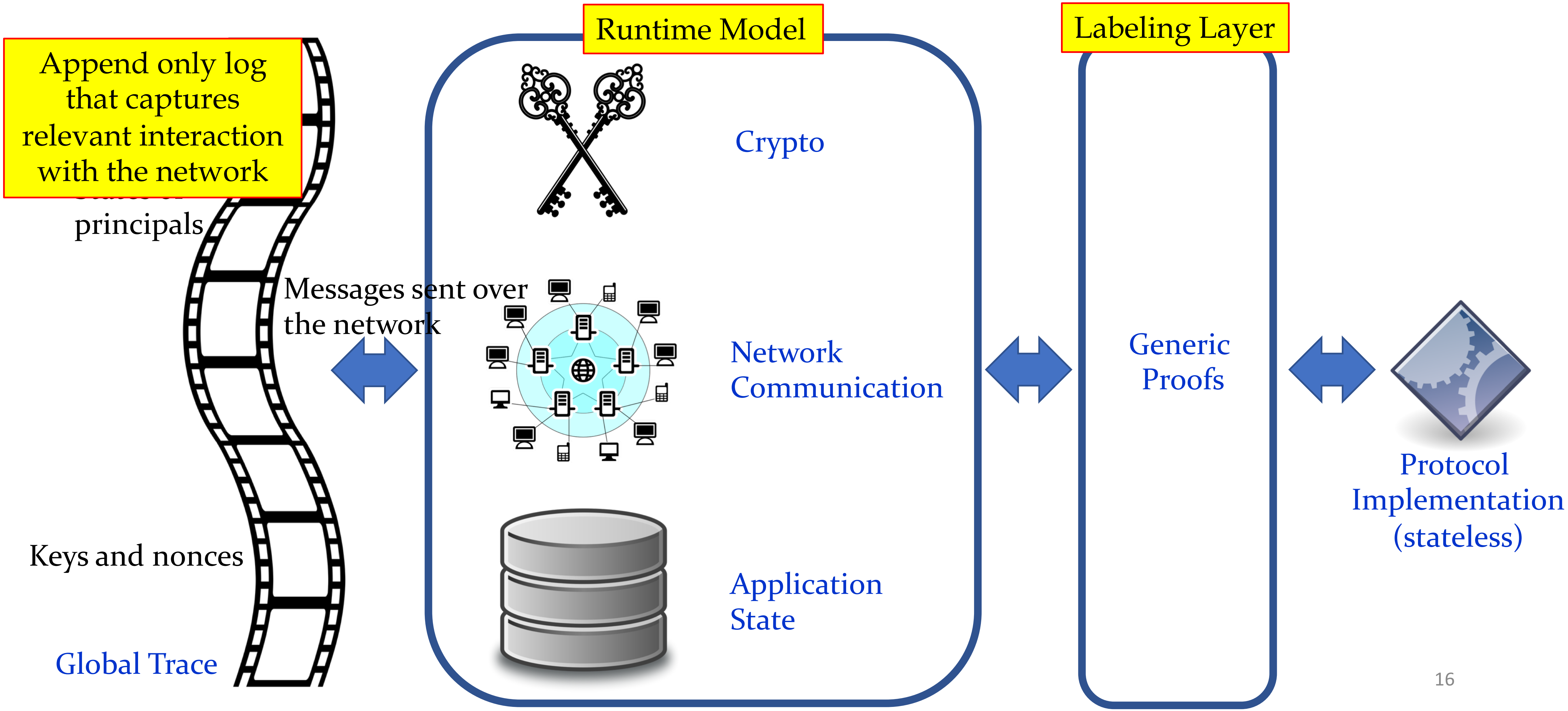
modular proofs
implementation level analysis
unbounded structures
inductive reasoning
executable models
interoperability

missing global view
limited expressivity w.r.t. security prop.
limited support for mutable state
less automation
no equational theories (e.g., DH)

```
{
  i0 = (uint32_t)32U;
}
uint8_t *nkey = key_block;
if (key_len <= (uint32_t)64U)
{
  memcpy([illegible], key, key_len * sizeof (uint8_t));
}
else
{
  EverCrypt_Hash_hash_256(key, key_len, nkey);
}
KRML_CHECK_SIZE(sizeof ([illegible]8_t), l);
uint8_t ipad[l];
memset(ipad, (uint8_t)0x36U, l * sizeof (uint8_t));
for (uint32_t i = (uint32_t)0U; i < l; i++)
{
  uint8_t xi = ipad[i];
  uint8_t yi = key_block[i];
  ipad[i] = xi ^ yi;
}
KRML_CH[illegible]ZE(sizeof (uint8_t), l);
uint8_t [illegible][l];
memset(opad, (uint8_t)0x5cU, l * sizeof (uint8_t));
for (uint32_t i = (uint32_t)0U; i < l; i++)
{
  uint8_t xi = opad[i];
  uint8_t yi = key_block[i];
  opad[i] = xi ^ yi;
}
uint32_t
scrut[8U] =
  {
    (uint32_t)0x6a09e667U, (uint32_t)0xbb67ae85U, (uint32_t)[illegible]ef372U, (uint32_t)0xa54ff53aU,
    (uint32_t)0x510e527fU, (uint32_t)0x9b05688cU, (uint32_t)[illegible]3d9abU, (uint32_t)0x5be0cd19U
  };
uint32_t *s[illegible]crut;
uint8_t *d[illegible] = ipad;
Hacl_Hash_Core_SHA2_init_256(s);
if (data_len == (uint32_t)0U)
{
  EverCrypt_Hash_update_last_256(s, (uint64_t)0U, ipad, (uint32_t)64U);
}
```

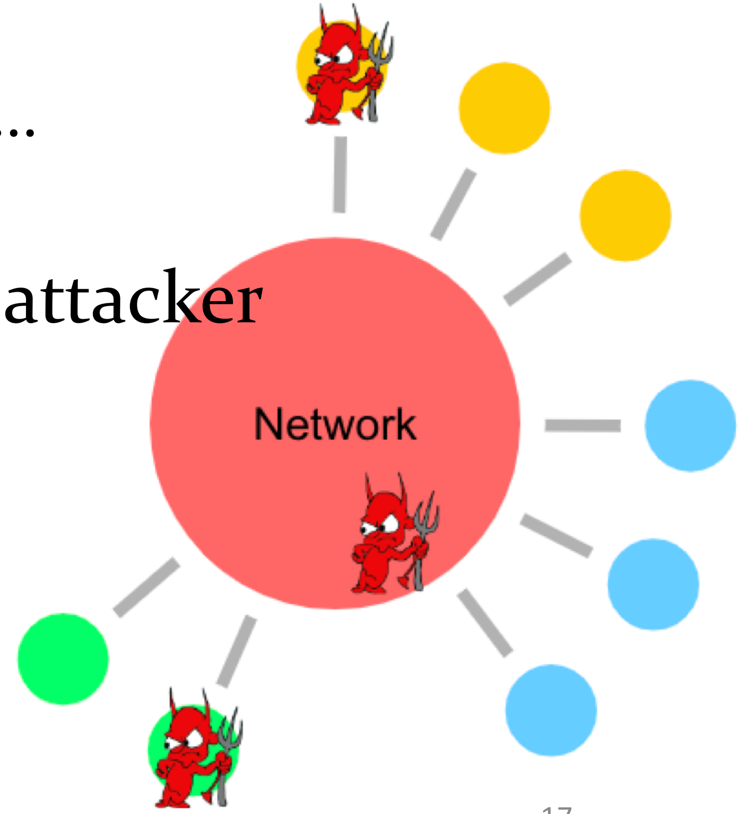
F*

DY* Architecture

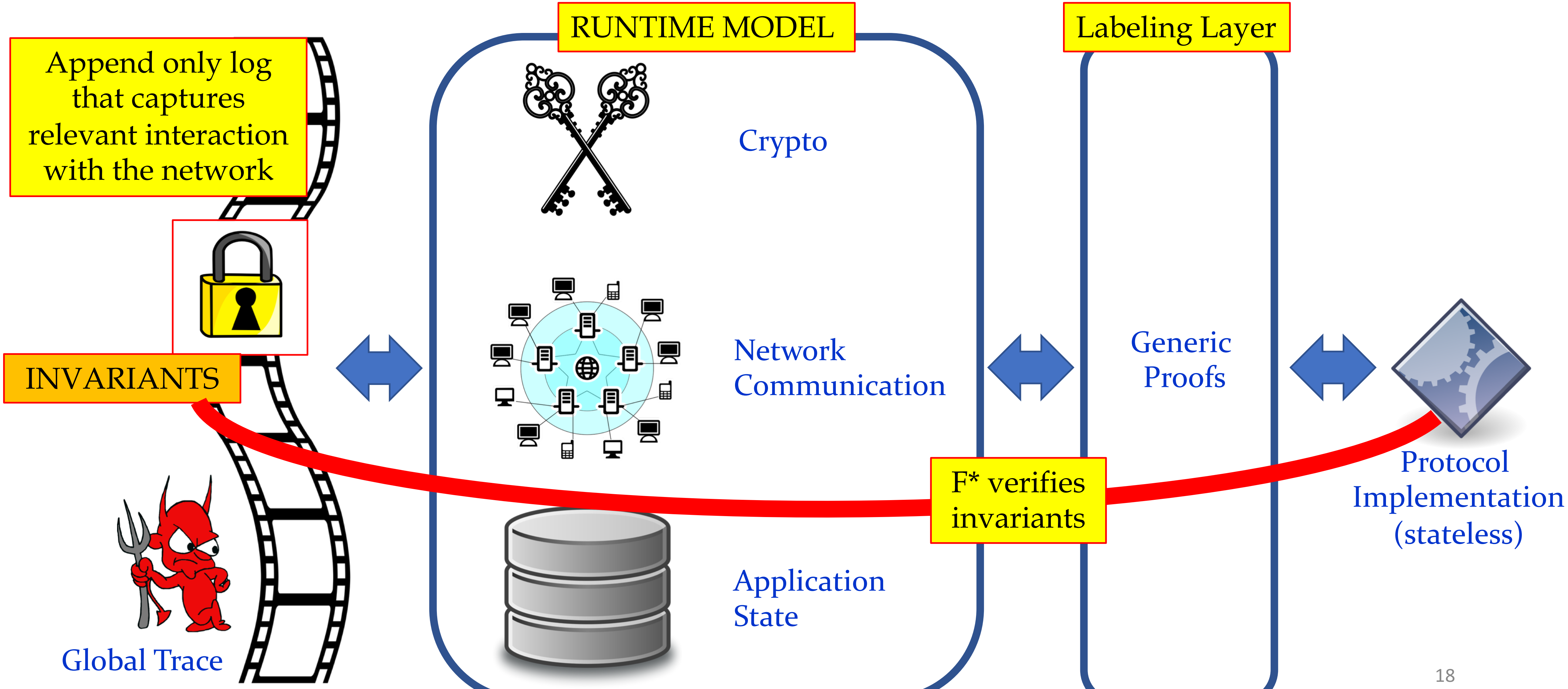


Attacker Model

- Active Network Attacker
  - Can derive arbitrary messages from its knowledge
  - Cannot "break" crypto,
    i.e., no decryption w/o key, no forging of signatures, …
  - Can (dynamically) corrupt principals
- Goal: Show that protocol is secure given such an attacker



DY* Architecture



Security Properties in DY*

- Forward secrecy

```
val initiator_forward_secrecy_lemma:
      i:timestamp -> a:principal -> b:principal ->
      gx:bytes -> gy:bytes -> k:bytes -> LCrypto unit (pki isodh)

 (requires (fun t0 -> i < trace_len t0 /\
                did_event_occur_at i a (finishI a b gx gy k)))

 (ensures (fun t0 _ t1 -> t0 == t1 /\ (
        corrupt_at i (P b) \/
        (exists si sj vi vj . is_labeled isodh_global_usage i k
                    (join (readers [V a si vi]) (readers [V b sj vj])) /\
                    ( corrupt_at (trace_len t0) (V a si vi) \/
                     corrupt_at (trace_len t0) (V b sj vj) \/
                     is_unknown_to_attacker_at (trace_len t0) k)
 ))))
```

Whenever Alice finishes the protocol s.t. Alice assumes that she talked to Bob and exchanged a key k ...

... then Bob was compromised during the protocol run OR ...

... the exchanged key k is unknown to the attacker.

Signal Messaging Protocol in DY*

- 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
- Appeared at IEEE European Symposium on Security and Privacy (EuroS&P 2021)

Signal Messaging Protocol in DY*

- First mechanized proof accounting for
  - Forward Secrecy
  - Post-compro
  - Unbounded r
    rounds at the
- First type-base
  post-comprom

	Modules	FLoC	PLoC	Verif. Time
Generic DY*	9	1,536	1,344	$\approx$ 3.2 min
NS-PK	4	439	-	(insecure)
NSL	5	340	188	$\approx$ 0.5 min
ISO-DH	5	424	165	$\approx$ 0.9 min
ISO-KEM	4	426	100	$\approx$ 0.7 min
Signal	8	836	719	$\approx$ 1.5 min

- First analysis of Signal based on dependent types
- Appeared at IEEE European Symposium on Security and Privacy (EuroS&P 2021)

Noise : Family of 59+ protocols

FAST, MODERN, SECURE VPN TUNNEL

WhatsApp

IKpsk2:	**IK:**	**XK:**
$\leftarrow$ s	$\leftarrow$ s	$\leftarrow$ s
...	...	...
$\rightarrow$ e, es, s, ss	$\rightarrow$ e, es, s, ss	$\rightarrow$ e, es
$\leftarrow$ e, ee, se, psk	$\leftarrow$ e, ee, se	$\leftarrow$ e, ee
		$\rightarrow$ s, se

How do we verify so many protocols?

Noise* Security Analysis

- Previous works do not cover important details like message formats, protocol state machines, or key management.
- Perform per-instance verification of each Noise protocol
- For new protocols derived from the framework, the verification needs to be performed again with the implementation done

- Our approach does this analysis in a generic manner, once and for all
  - Even valid for future protocols
  - Appeared at IEEE Symposium on Security and Privacy (Oakland) this year

Future Directions and Collaborations

- Lots of interesting work to be done!
  - Group messaging protocols
    - Novel security properties
    - Have not been tried with existing tools
  - Concrete DY*: fully verified implementations
    - Plug-and-play reference implementations
  - Equivalence properties
  - WIM*: mechanize the **Web Infrastructure Model**
  - Contact me at abhishek.b@iitgn.ac.in if you are interested in collaborating on this

Inria INVENTEURS DU MONDE NUMÉRIQUE | University of Stuttgart Institute of Information Security | Microsoft | ROYAL HOLLOWAY UNIVERSITY OF LONDON | INDIAN INSTITUTE OF TECHNOLOGY DELHI