University of Stuttgart
Institute of
Information Security

Carnegie Mellon University

Inria

Indian Institute of Technology Gandhinagar

DY*: A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code [EuroS&P 21]

GDR'21 | sec.uni-stuttgart.de

SEC

Bhargavan, Bichhawat, Do, Hosseyni, Küsters, Schmitz, Würtele

Cryptographic Protcols 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 Protcols are Everywhere: Golden Era of Crypto

COMPUTERWORLD UNITED STATES INSIDER

NEWS

EMV flaw allows 'pre-play' attacks on chip-enabled payment cards

Cambridge university researchers find weaknesses in the EMV protocol that can facilitate cloning-like attacks for chip-and-PIN payment cards

- Payment: EMV, W3C Web Payments, ...
- Post-Quantum Crypto: NIST KEMs, Signature, ...
- Lightweight Crypto: IETF LAKE, NIST LWC

QUIC

Wi Fi™

SAML v2.0 Security Assertion Markup Language

Let's Encrypt

Signal

zigbee

OAUTH 2

Cryptographic Protcols are Everywhere: Golden Era of Crypto

COMPUTERWORLD UNITED STATES

NEWS

EMV flaw allows 'pre payment cards

Cambridge university researchers find weaknesses in the EMV protocol that can facilitate cloning-like attacks for chip-and-PIN payment cards

ZDNet CENTRAL EUROPE MIDDLE EAST SCANDINAVIA AFRICA UK ITALY SPAIN MORE NEWSLETTERS

'Triple handshake' bug another big problem for TLS/SSL

- Payment: EMV, W3C Web Payments, ...
- Post-Quantum Crypto: NIST KEMs, Signature, ...
- Lightweight Crypto: IETF LAKE, NIST LWC

Cryptographic Protcols are Everywhere: Golden Era of Crypto

COMPUTERWORLD UNITED STATES

NEWS

EMV flaw allows 'pre payment cards

ZDNet

CENTRAL EUROPE MIDDLE EAST SCANDINAVIA AFRICA UK ITALY SPAIN MORE NEWSLETTERS

'Triple handshake' bug another big problem for TLS/SSL

CLOUDFLARE The Cloudflare Blog

Product News Speed & Reliability Security Serverless Zero Trust Developers Deep Dive Life @Cloudflare

Logjam: the latest TLS vulnerability explained

21/05/2015

UIC

SAML v2.0 Security Assertion Markup Language

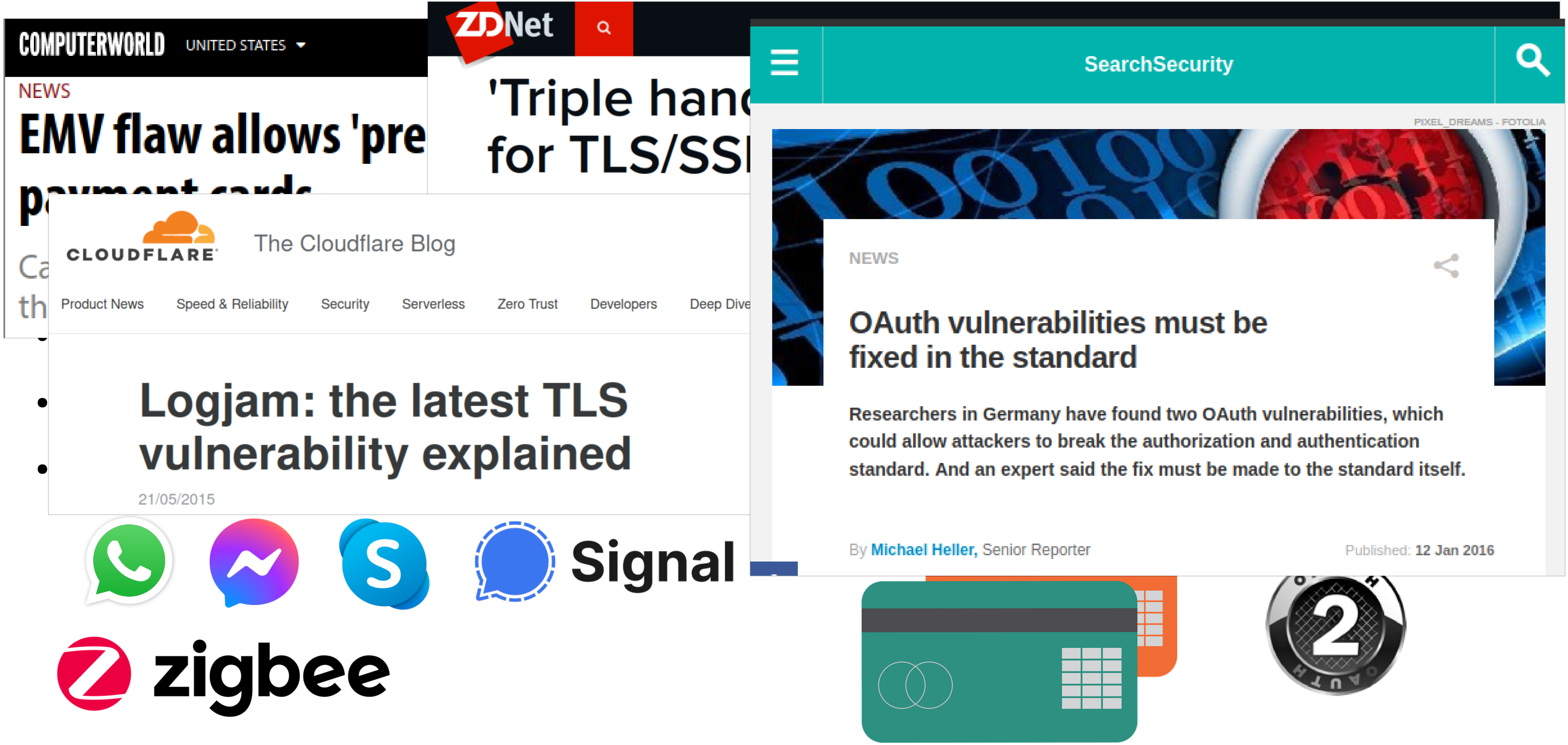
Let's Encrypt

Signal

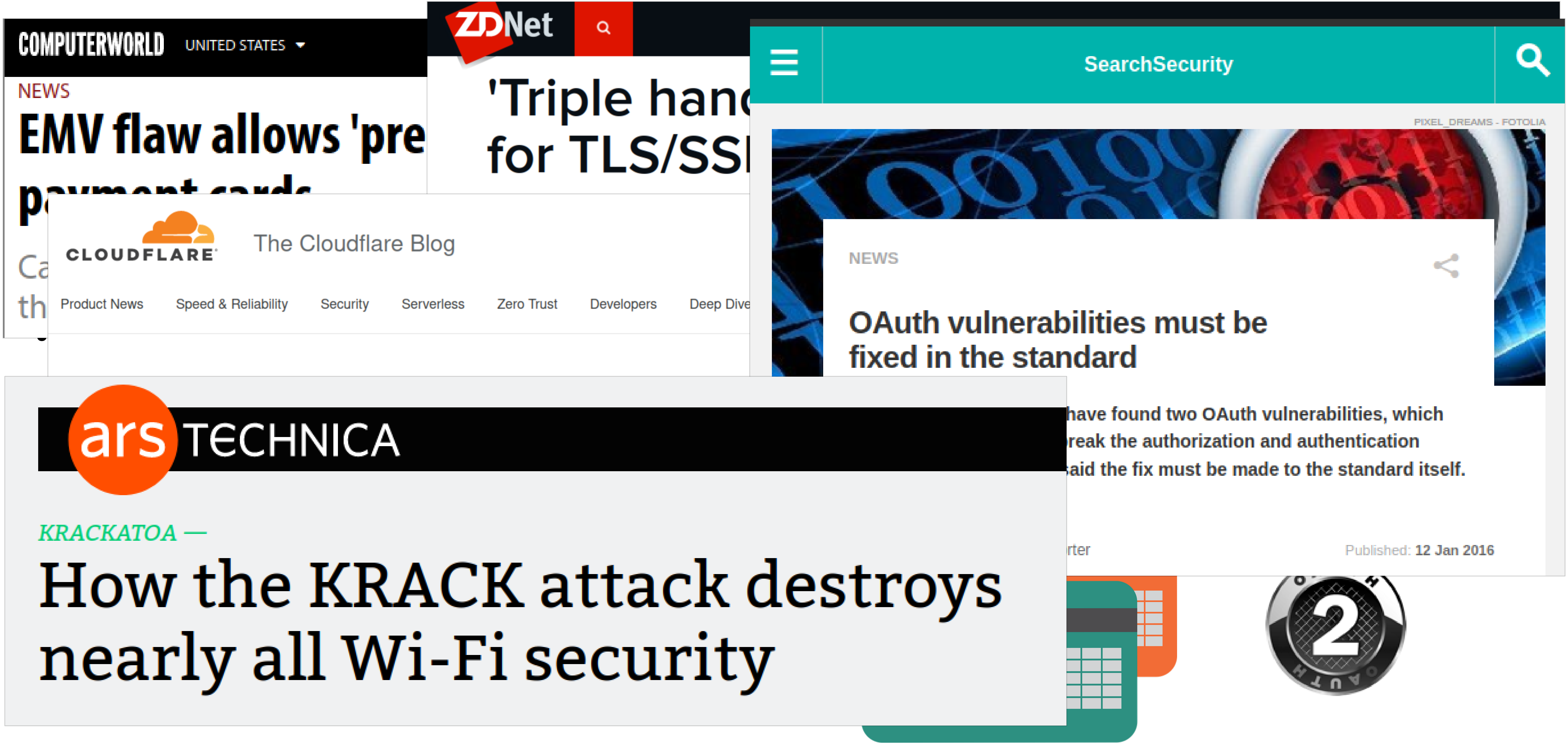
zigbee

OAUTH 2

Cryptographic Protcols are Everywhere: Golden Era of Crypto



Cryptographic Protcols are Everywhere: Golden Era of Crypto

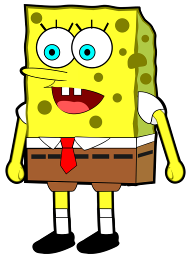


Signal Messaging Protocol

- Asynchronous continuous key exchange protocol

Signal

$G = \langle g \rangle$



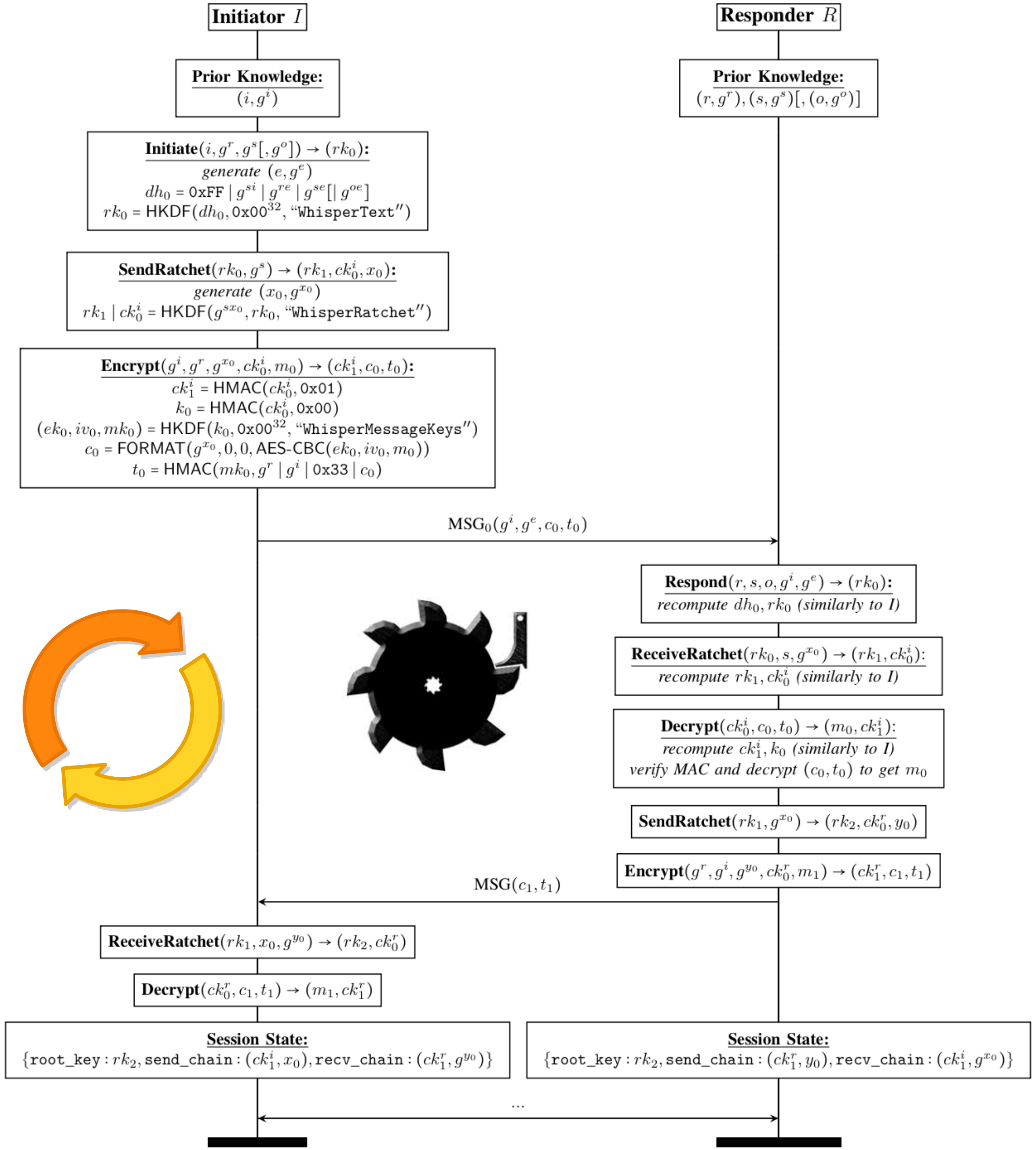
x random $\xrightarrow{g^x}$

$(g^y)^x = g^{xy} \xleftarrow{g^y}$ y random

$(g^x)^y = g^{xy}$

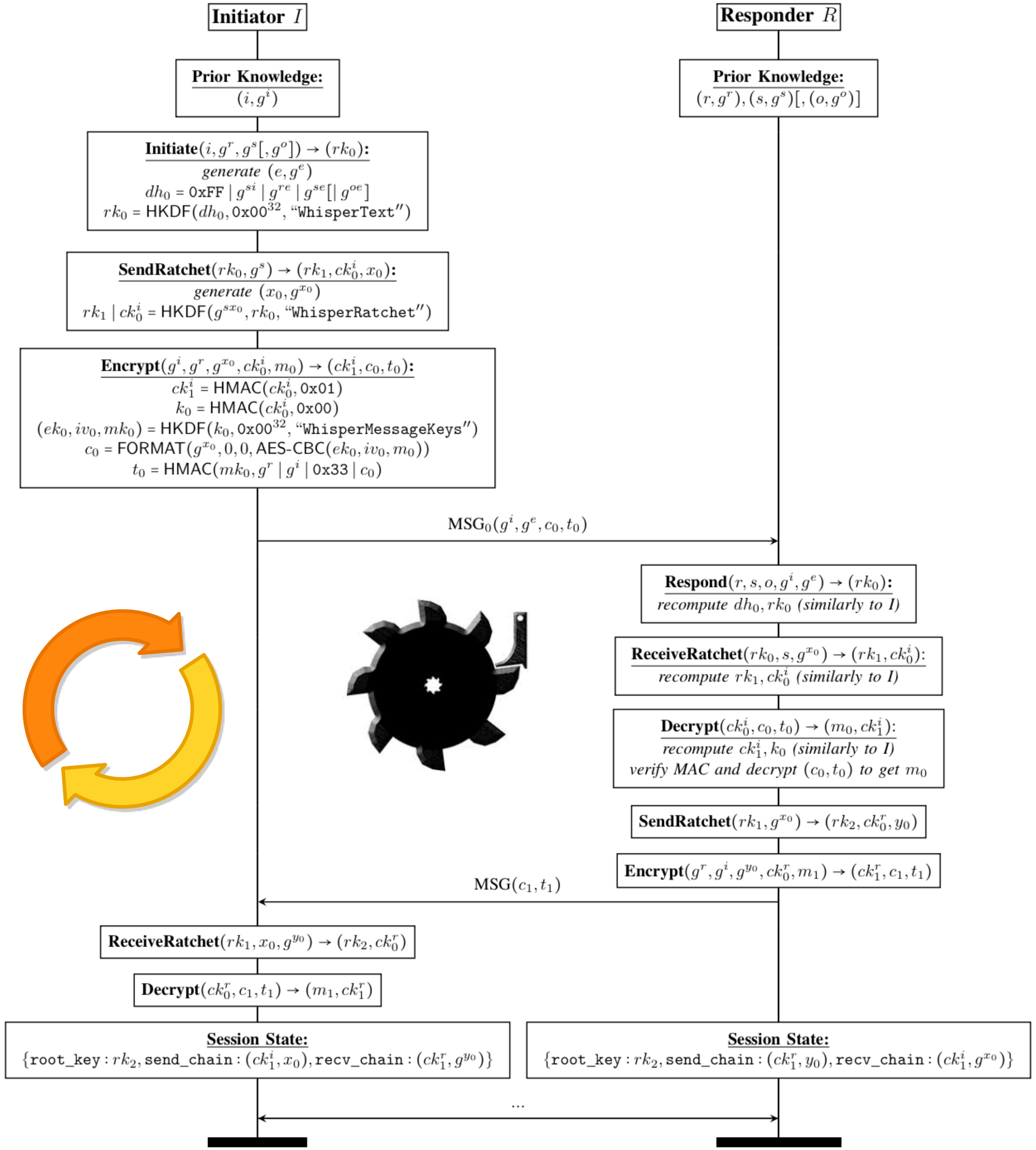
Signal Messaging Protocol

- Asynchronous continuous key exchange protocol



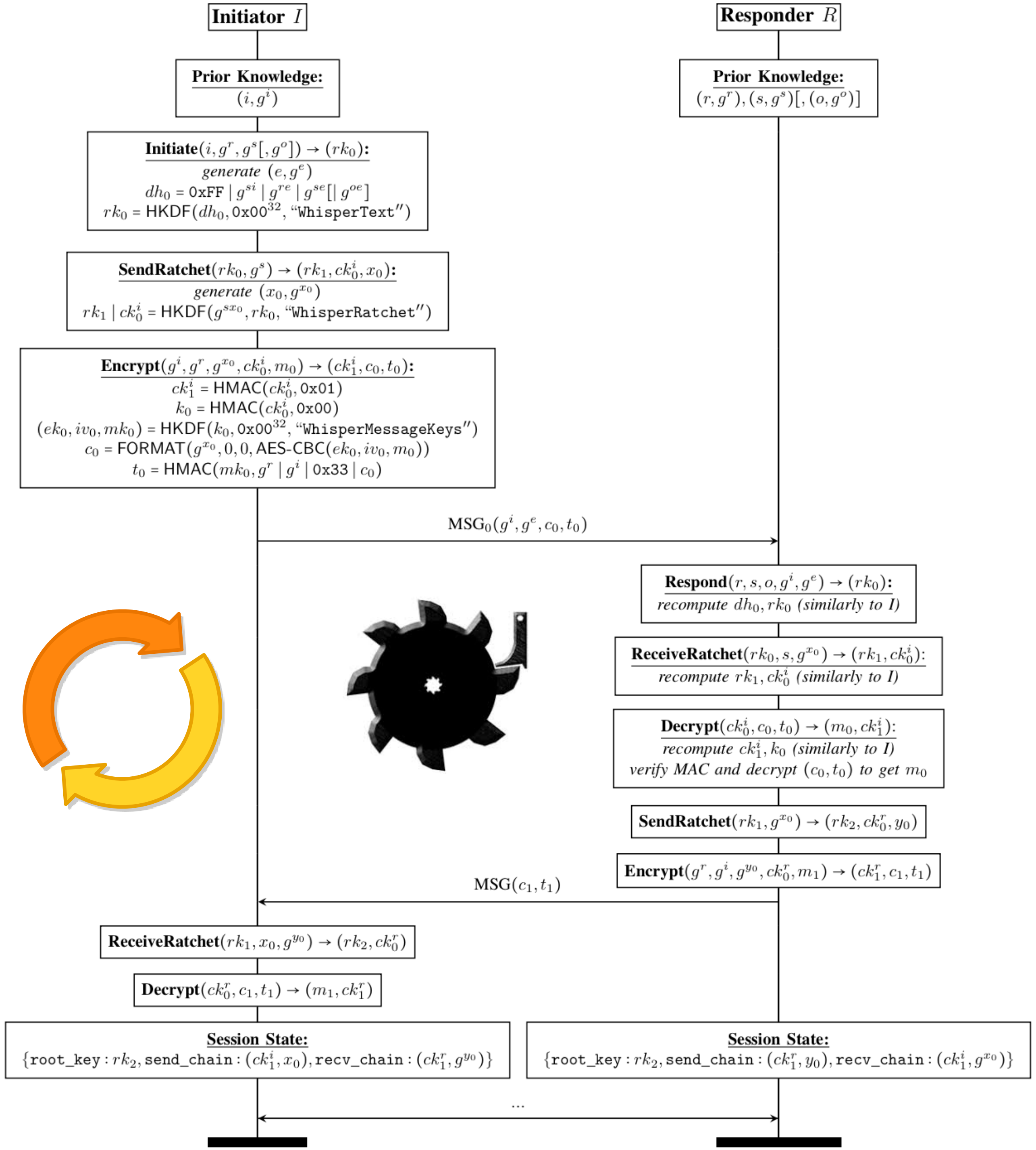
Signal Messaging Protocol

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



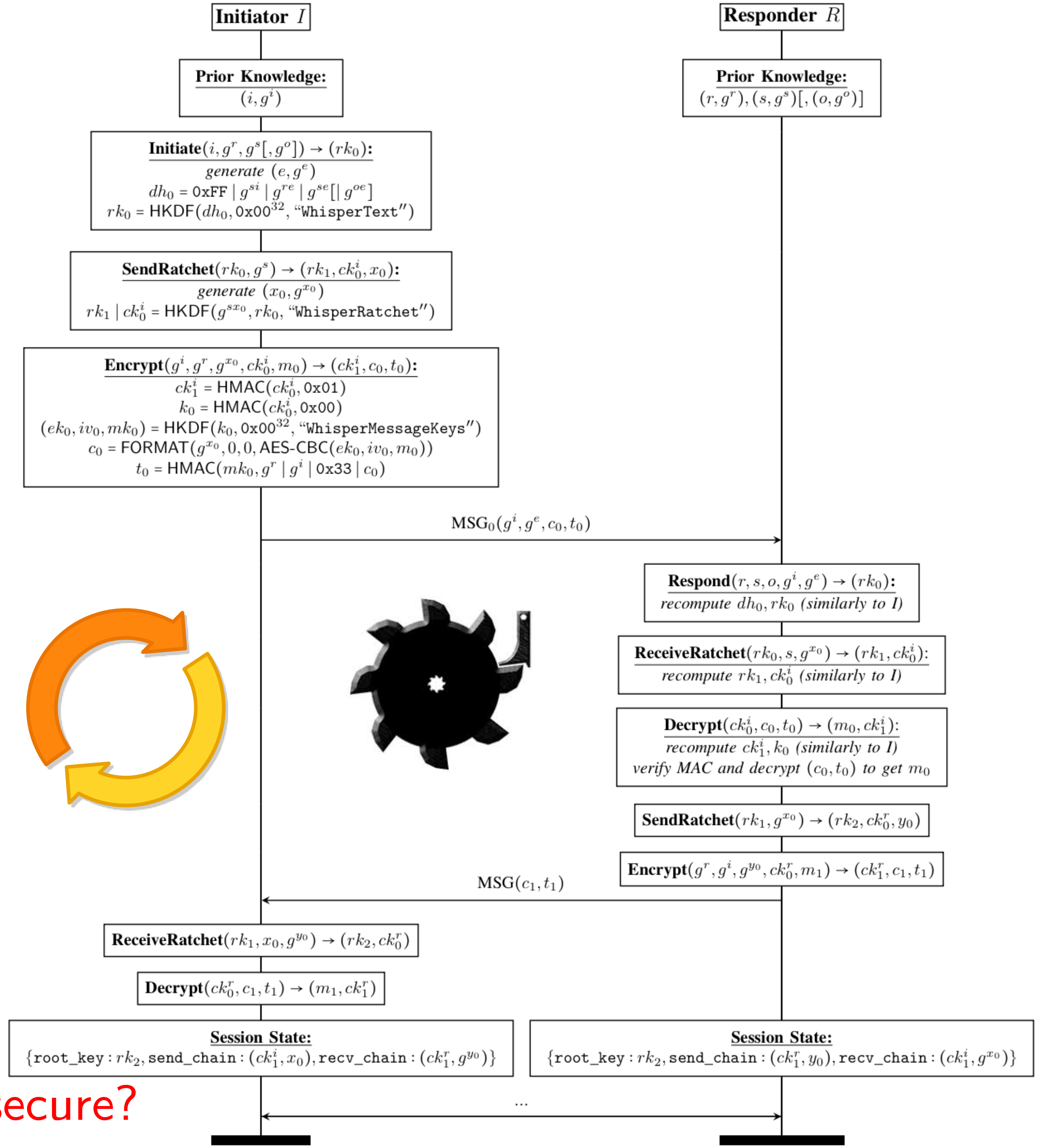
Signal Messaging Protocol

- 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

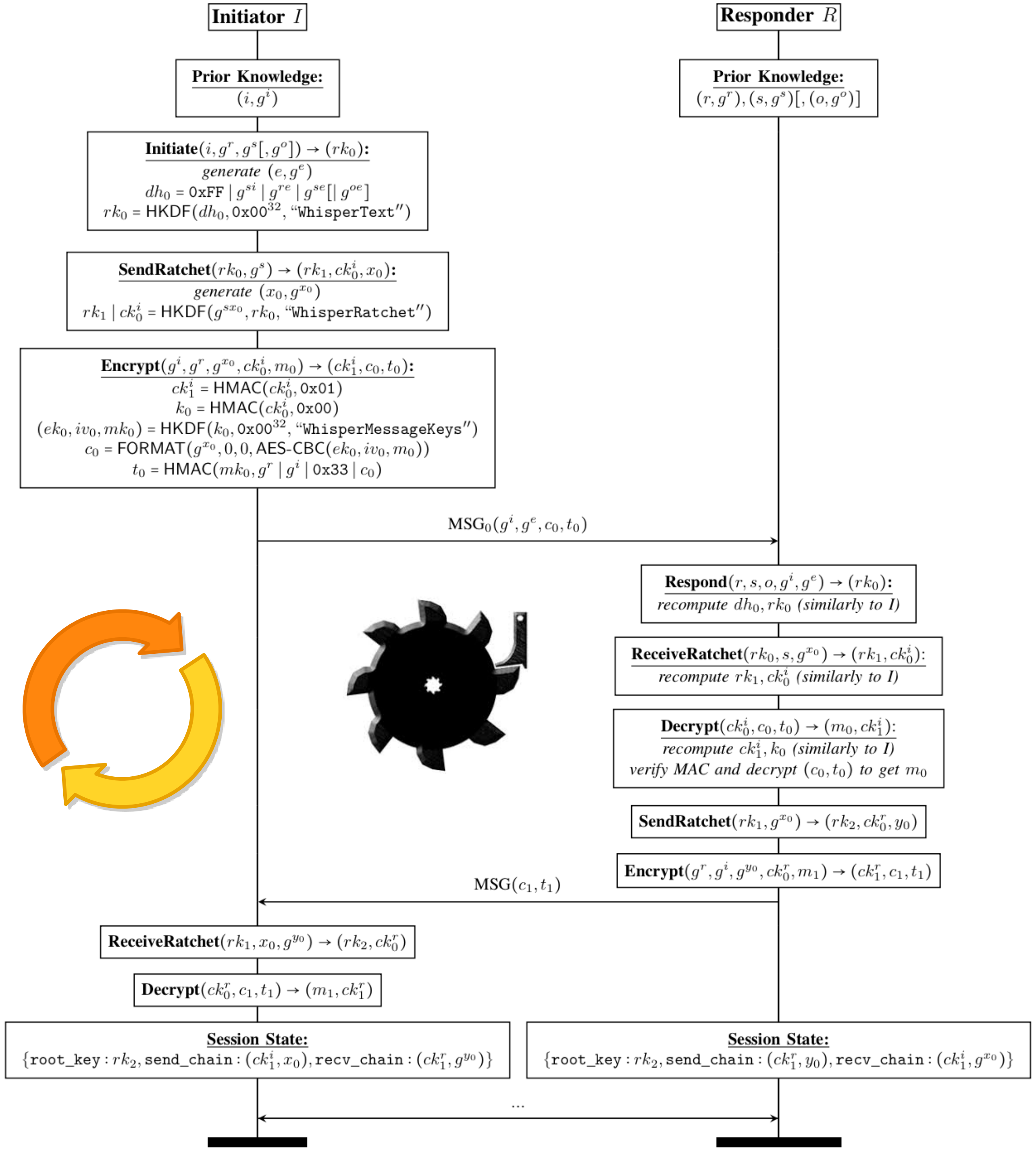


Signal Messaging Protocol

- 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
- Can we mechanically verify that the protocol is secure?

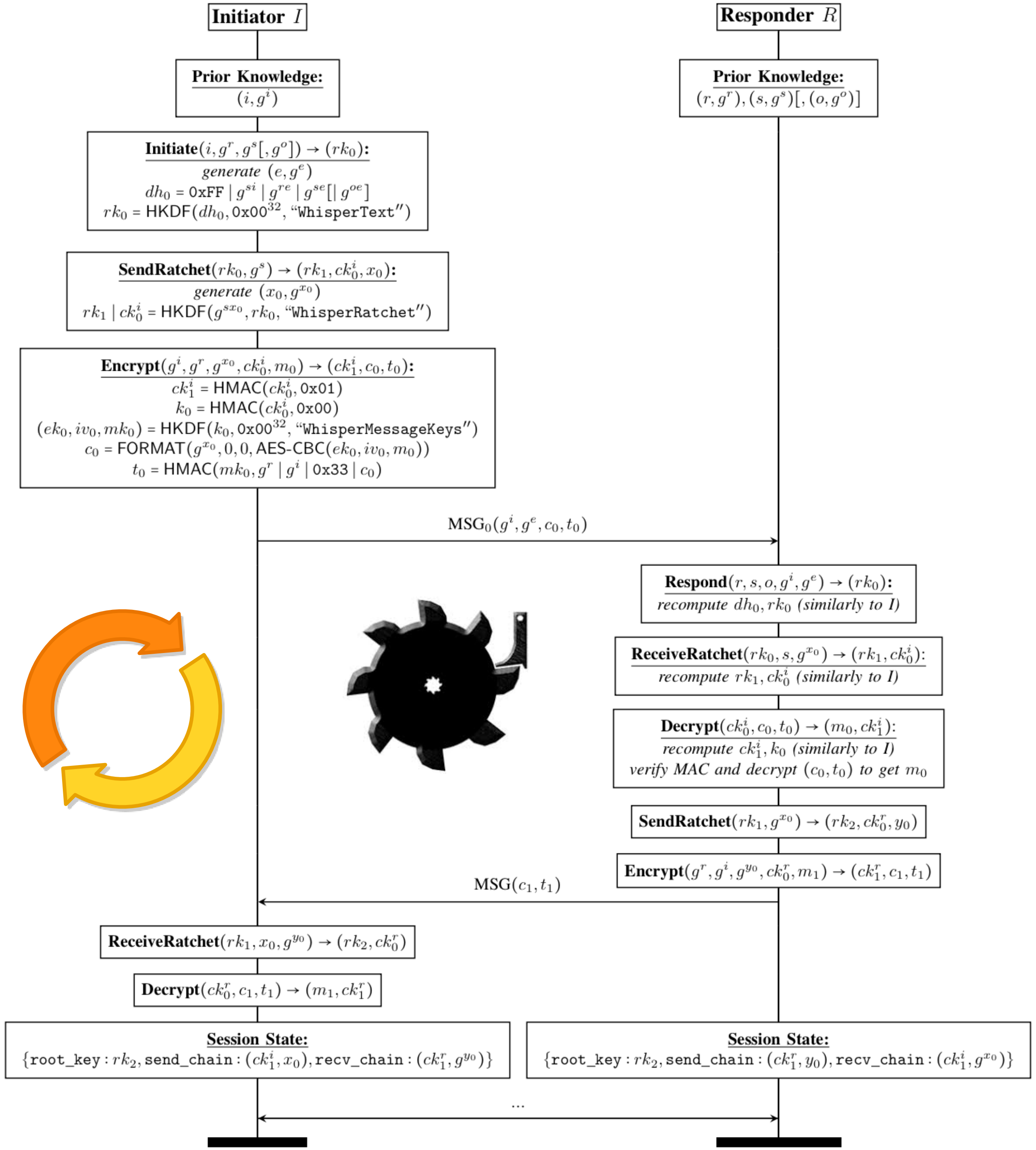


Formalizing Signal



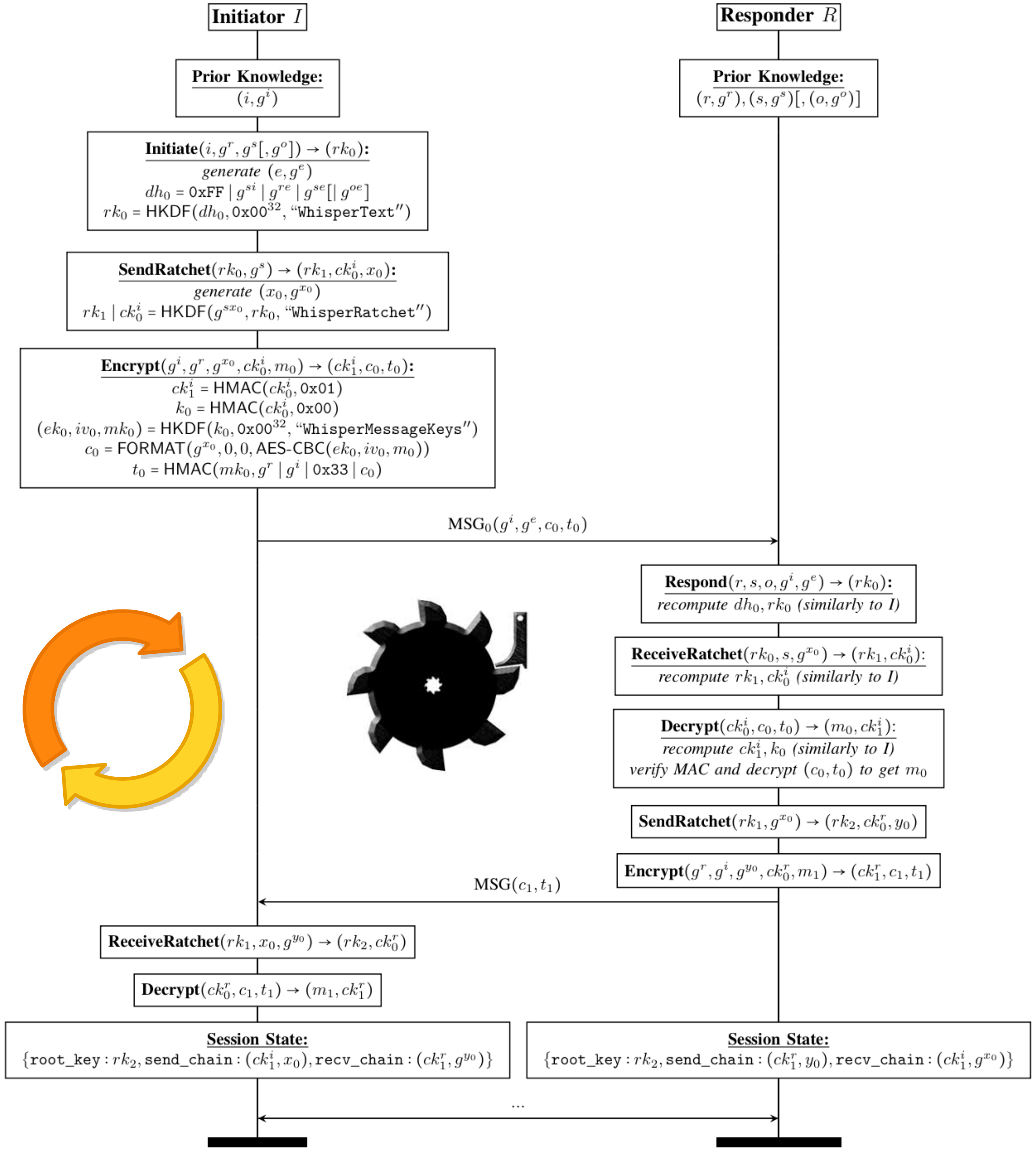
Formalizing Signal

- 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

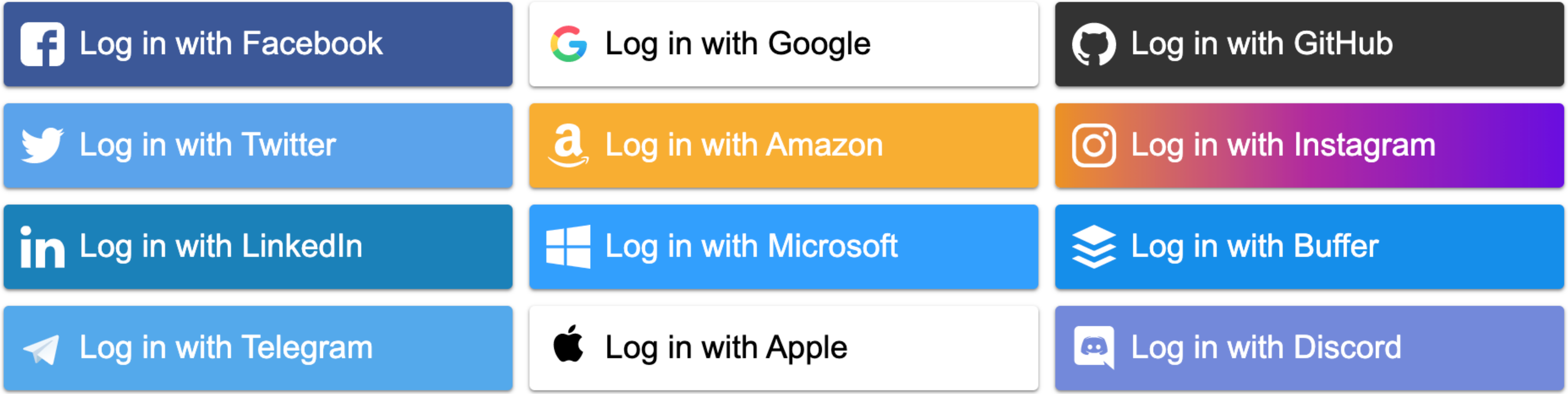


Security Protocols: SSO

Single Sign-On allows users to authenticate to service A using an account at service B

Security Protocols: SSO

Single Sign-On allows users to authenticate to service A using an account at service B



Security Protocols: SSO

Single Sign-On allows users to authenticate to service A using an account at service B

- One account for many services

Security Protocols: SSO

Single Sign-On allows users to authenticate to service A using an account at service B

- One account for many services
- Widely adopted: Web, IoT, Enterprise Networks, …

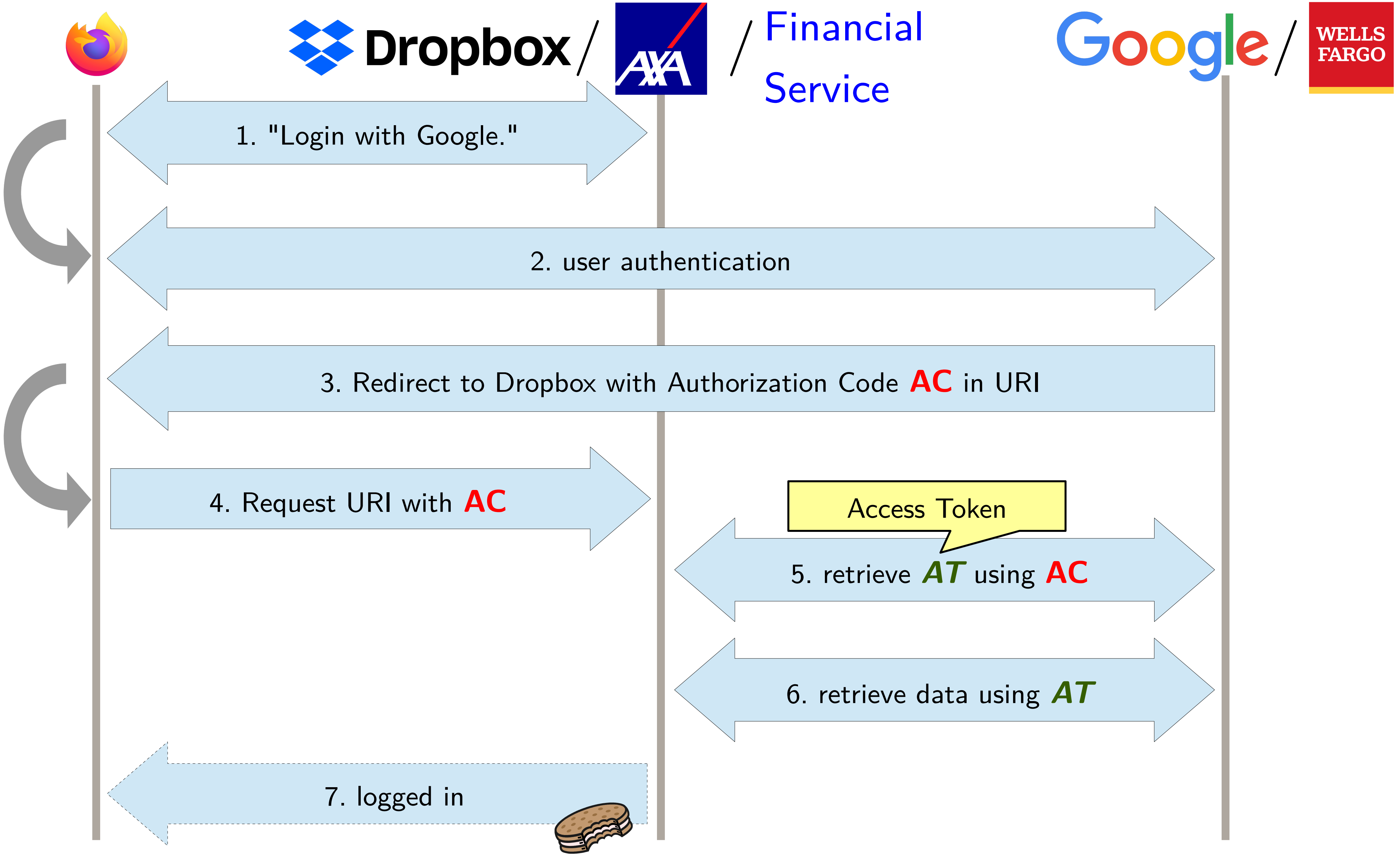
Security Protocols: SSO

Single Sign-On allows users to authenticate to service A using an account at service B

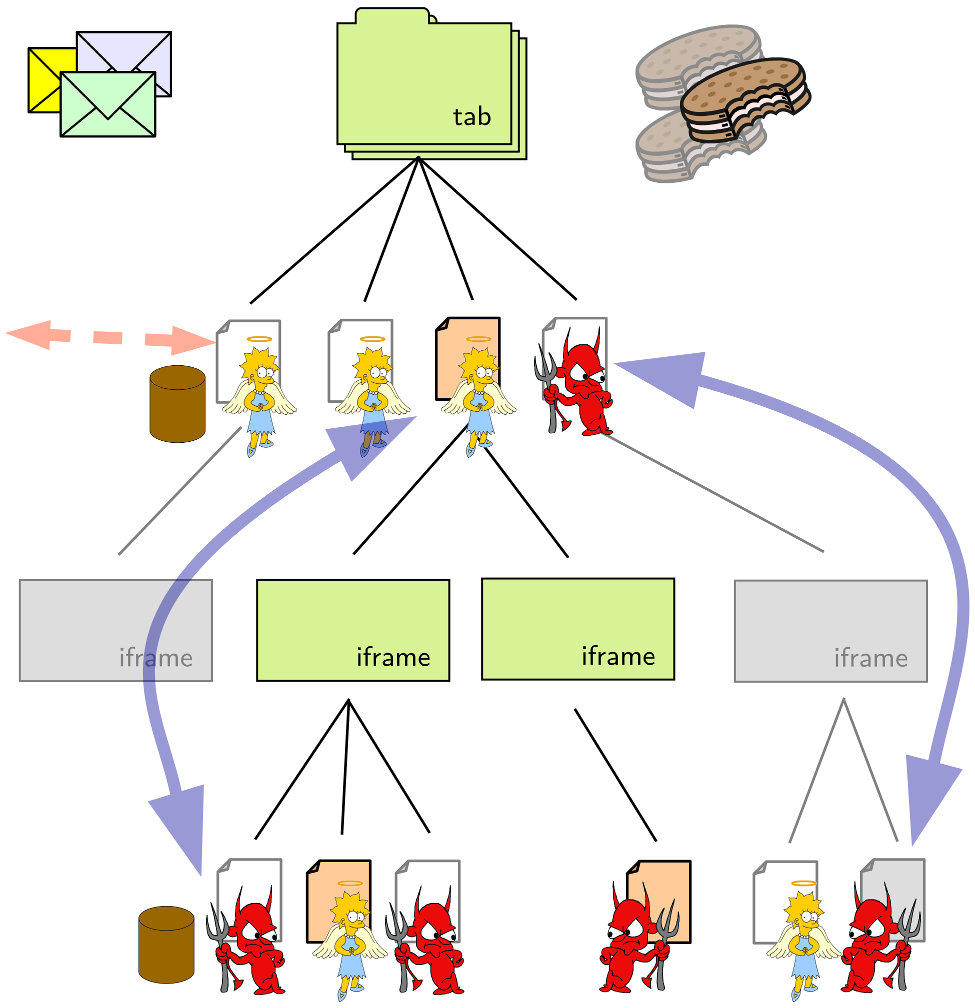
- One account for many services
- Widely adopted: Web, IoT, Enterprise Networks, …

OAuth2.0

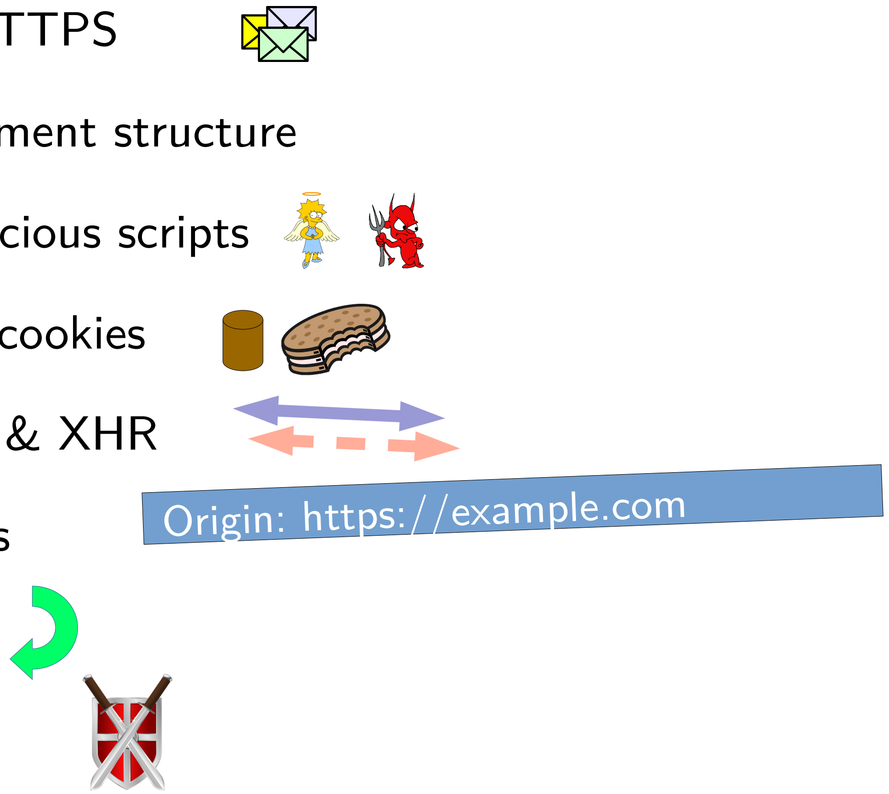
Authorization Code Mode



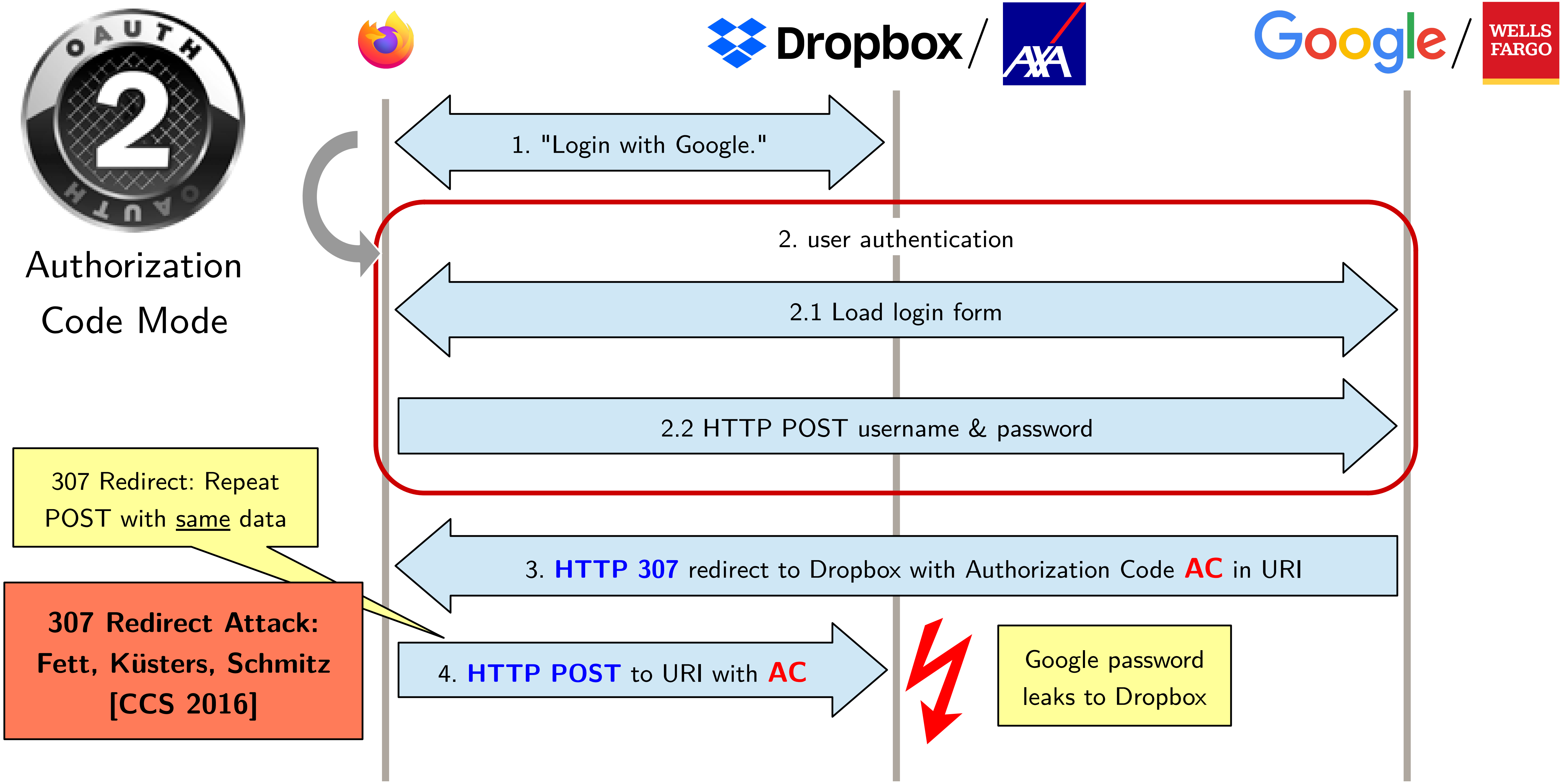
The Web is a Complex Environment



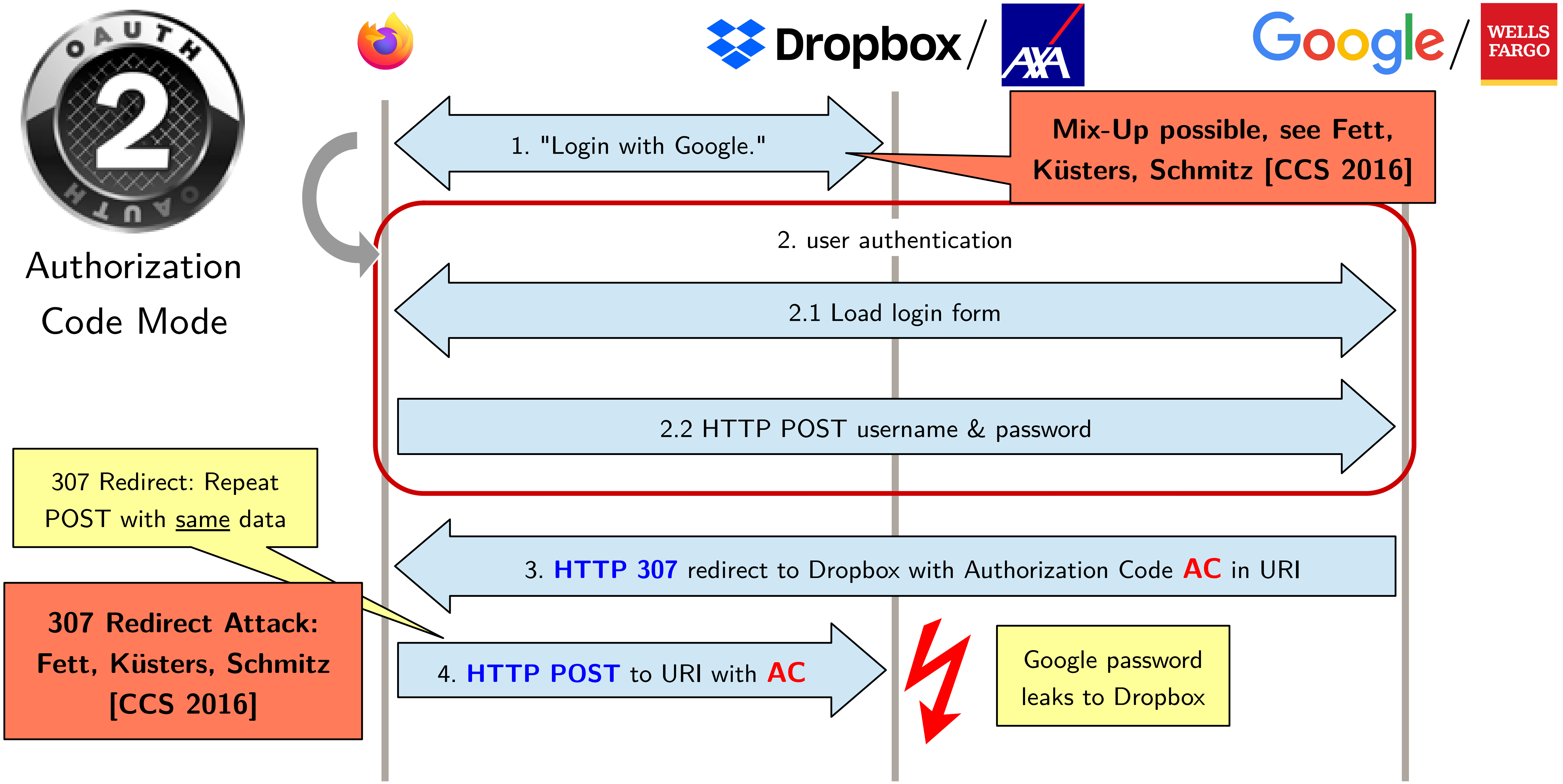
- DNS, HTTP, HTTPS
- window & document structure
- honest and malicious scripts
- Web storage & cookies
- Web messaging & XHR
- message headers
- redirections
- security policies
- ...



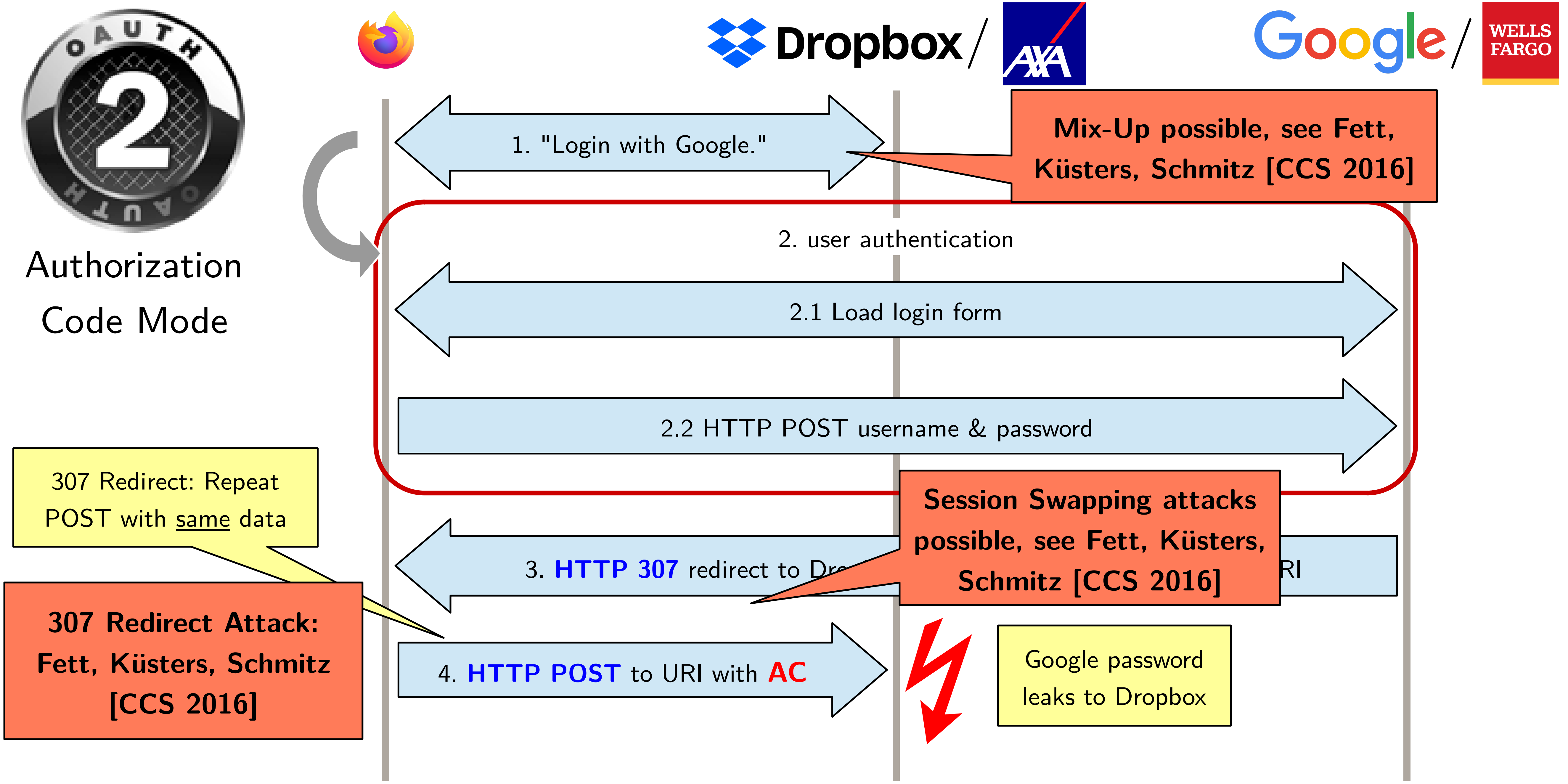
Many Attacks based on Web Features (Examples)



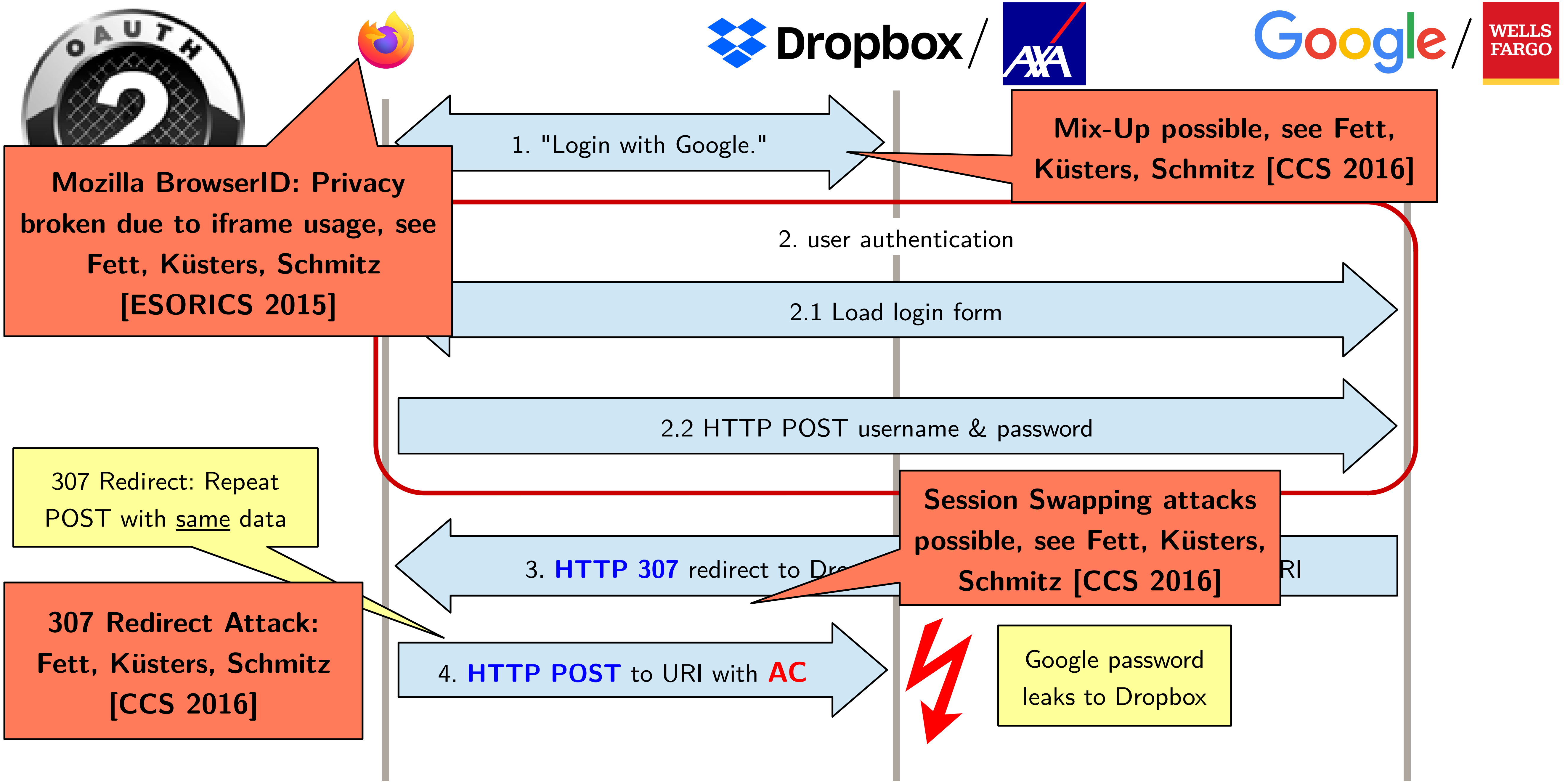
Many Attacks based on Web Features (Examples)



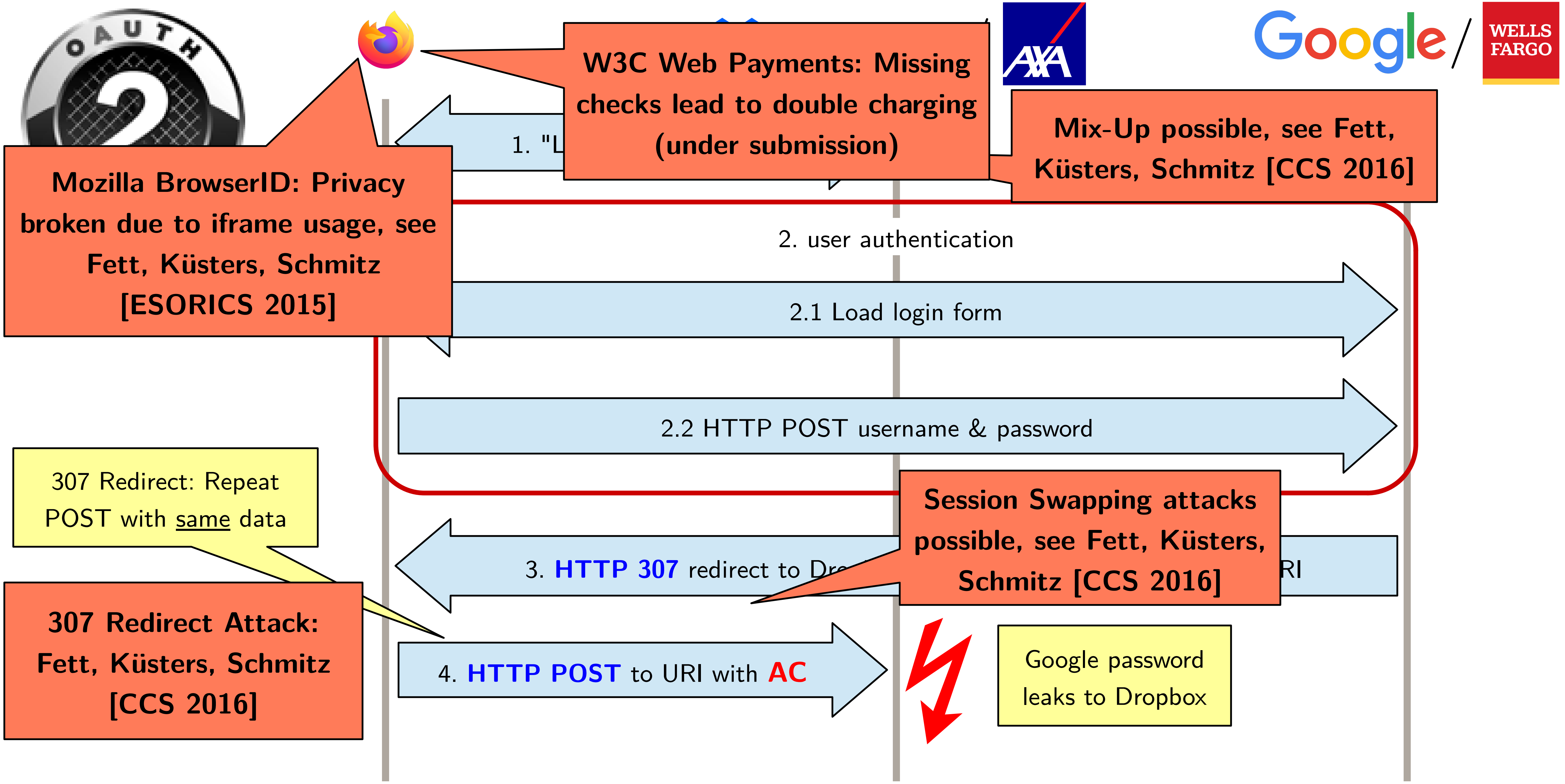
Many Attacks based on Web Features (Examples)



Many Attacks based on Web Features (Examples)



Many Attacks based on Web Features (Examples)



Analysis of Security Protocols on the Web: Main Research Lines

Analysis of Security Protocols on the Web: Main Research Lines

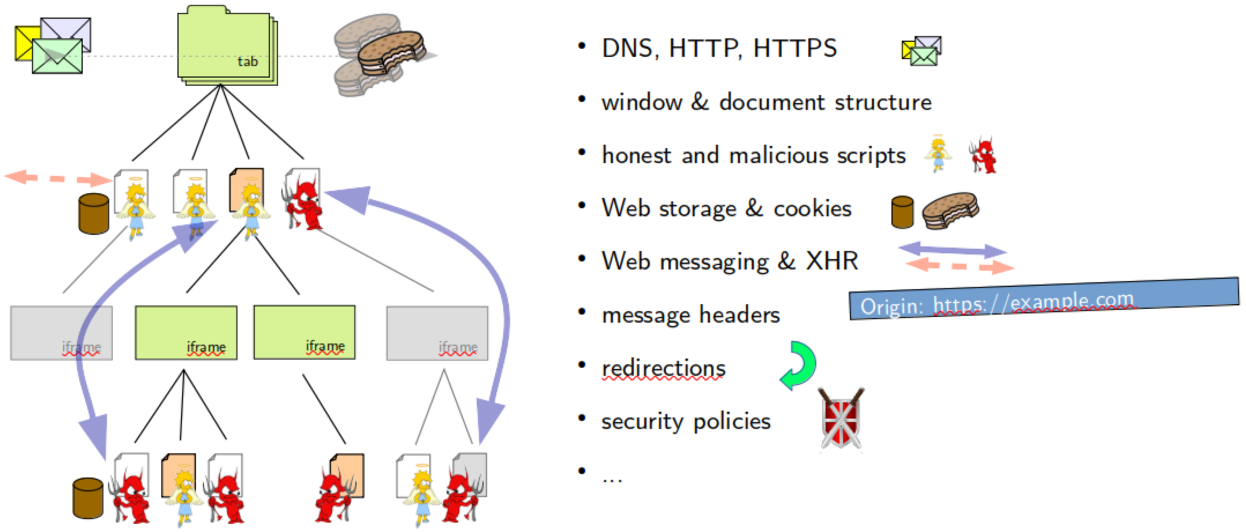
- Alloy
  - Finite-state model checker
  - Kerschbaum (2007), Akahwe et al. (2010), Pai et al (2011), Kumar (2012, 2014)
  - Very simplified and abstract models of the web

Analysis of Security Protocols on the Web: Main Research Lines

- Alloy
  - Finite-state model checker
  - Kerschbaum (2007), Akahwe et al. (2010), Pai et al (2011), Kumar (2012, 2014)
  - Very simplified and abstract models of the web
- ProVerif: WebSpi (Bansal et al., 2013)
  - Limitations imposed by ProVerif
  - Simplified and abstract models of the web

Analysis of Security Protocols on the Web: Main Research Lines

- Alloy
  - Finite-state model checker
  - Kerschbaum (2007), Akahwe et al. (2010), Pai et al (2011), Kumar (2012, 2014)
  - Very simplified and abstract models of the web
- ProVerif: WebSpi (Bansal et al., 2013)
  - Limitations imposed by ProVerif
  - Simplified and abstract models of the web
- Web Infrastructure Model (WIM)
  - Fett, Küsters, Schmitz [S&P 2014]
  - Detailed model
  - Used for many case studies (OAuth, OIDC, BrowserID, FAPI, W3C Web Payments)
  - Manual pen-and-paper analyses



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

- ✔ (mostly) automated analysis
- ✔ 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

- ✔ (mostly) automated analysis
- ✔ global trace & properties
- ✔ equational theories

Dependent Types:
RCF, F7, ...

focus on implementation aspects

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

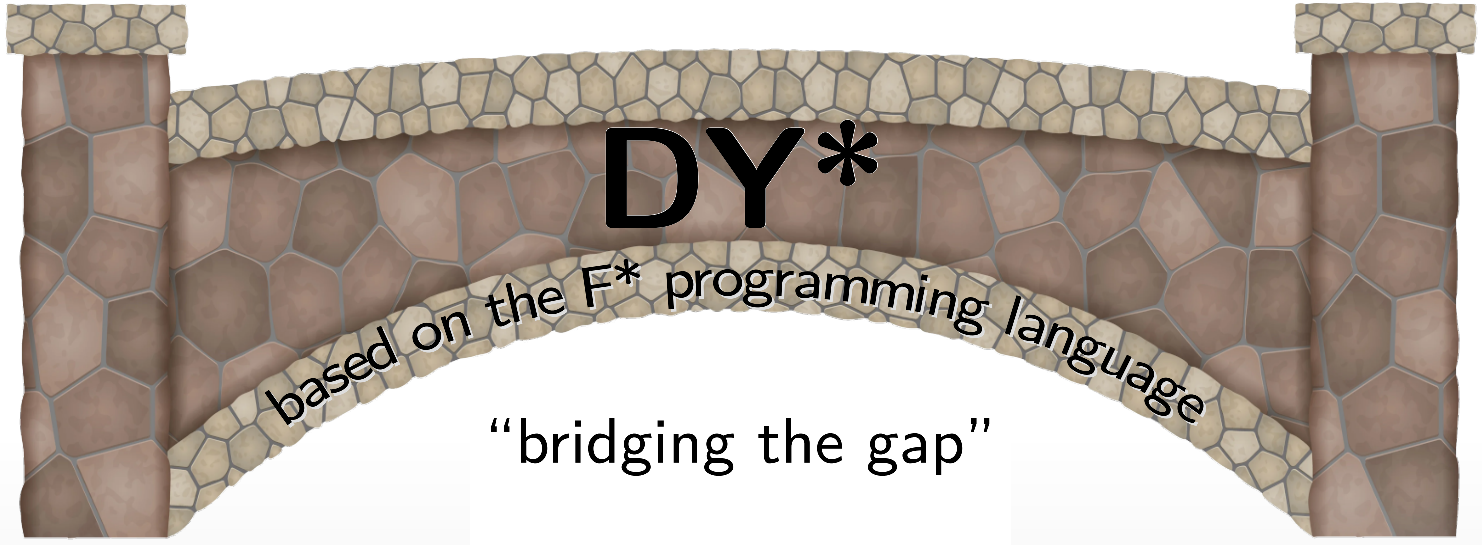
- ✗ missing global view
- ✗ limited expressivity w.r.t. security prop.
- ✗ limited support for mutable state
- ✗ less automation
- ✗ no equational theories (e.g., DH)

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



"bridging the gap"

- ✔ (mostly) automated analysis
- ✔ 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)

What is F*?

- Functional programming language aimed at program verification
- Developed and actively supported by Microsoft Research, INRIA, and others
- Already used to prove security of, for example, parts of TLS 1.3
- Rich, versatile type system
  - Dependent types
  - Refinement types
  - Pre/post conditions
  - Backed by SMT-Solver Z3
  - Can be used to precisely express strong (security) properties
- F* program can be translated to OCaml, F#, C, or JavaScript

F* - Simple Example

```
val factorial: nat -> nat

let rec factorial n =

    if n<=1 then 1 else n * (factorial (n-1))
```

F* - Simple Example

```
val factorial: nat -> nat

let rec factorial n =

    if n<=1 then 1 else n * (factorial (n-1))
```

Property:

$\forall\ n \in \mathbb{N}$. factorial $n > 0$

F* - Simple Example

```
val factorial: nat -> nat

let rec factorial n =

    if n<=1 then 1 else n * (factorial (n-1))
```

Property:

$\forall\ n \in \mathbb{N}$. factorial $n > 0$

```
val factorial_lemma: n:nat -> Lemma ( factorial n > 0 )
```

F* - Simple Example

```
val factorial: nat -> nat
let rec factorial n =
    if n<=1 then 1 else n * (factorial (n-1))
```

Property:
$\forall\ n \in \mathbb{N}$. factorial $n > 0$

```
val factorial_lemma: n:nat -> Lemma ( factorial n > 0 )
let rec  factorial_lemma n = match n with
    |0 -> ()
    |_ -> factorial_lemma (n-1)
```

F* - Simple Example

```
val factorial: nat -> nat

let rec factorial n =
    if n<=1 then 1 else n * (factorial (n-1))
```

Property:
$\forall\ n \in \mathbb{N}$. factorial $n > 0$

```
val factorial_lemma: n:nat -> Lemma ( factorial n > 0 )

let rec  factorial_lemma n = match n with
    |0 -> ()
    |_ -> factorial_lemma (n-1)
```

Induction Start

F* - Simple Example

```
val factorial: nat -> nat

let rec factorial n =

    if n<=1 then 1 else n * (factorial (n-1))
```

Property:
$\forall\ n \in \mathbb{N}$. factorial $n > 0$

```
val factorial_lemma: n:nat -> Lemma ( factorial n > 0 )

let rec  factorial_lemma n = match n with

    |0 -> ()

    |_ -> factorial_lemma (n-1)
```

Induction Start

Induction Step

F* - Simple Example

```
val factorial: nat -> nat
let rec factorial n =
    if n<=1 then 1 else n * (factorial (n-1))
```

Property:
$\forall\ n \in \mathbb{N}$. factorial $n > 0$

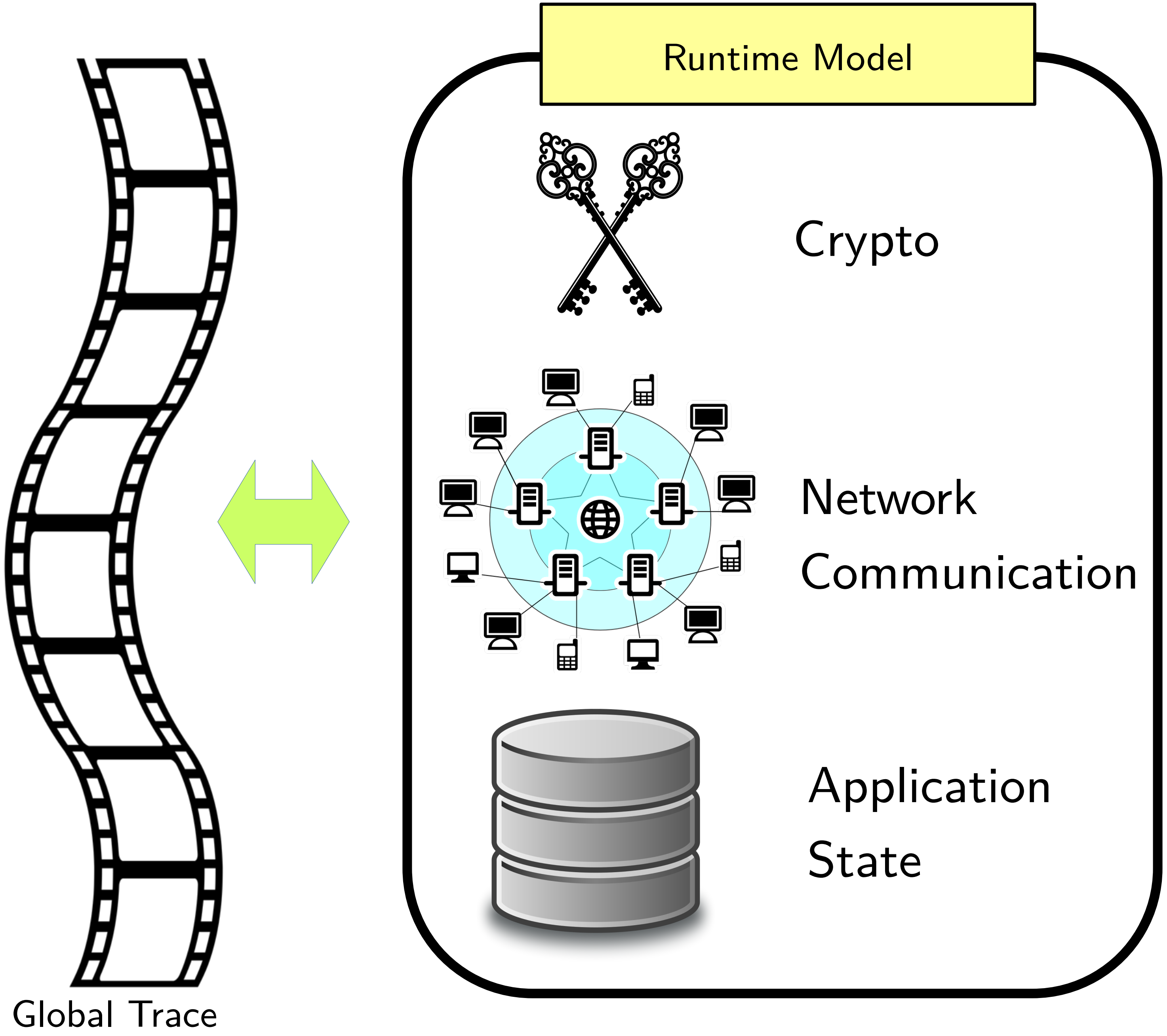
```
val factorial_lemma: n:nat -> Lemma ( factorial n > 0 )
let rec  factorial_lemma n = match n with
    |0 -> ()
    |_ -> factorial_lemma (n-1)
```

Induction Start

Induction Step

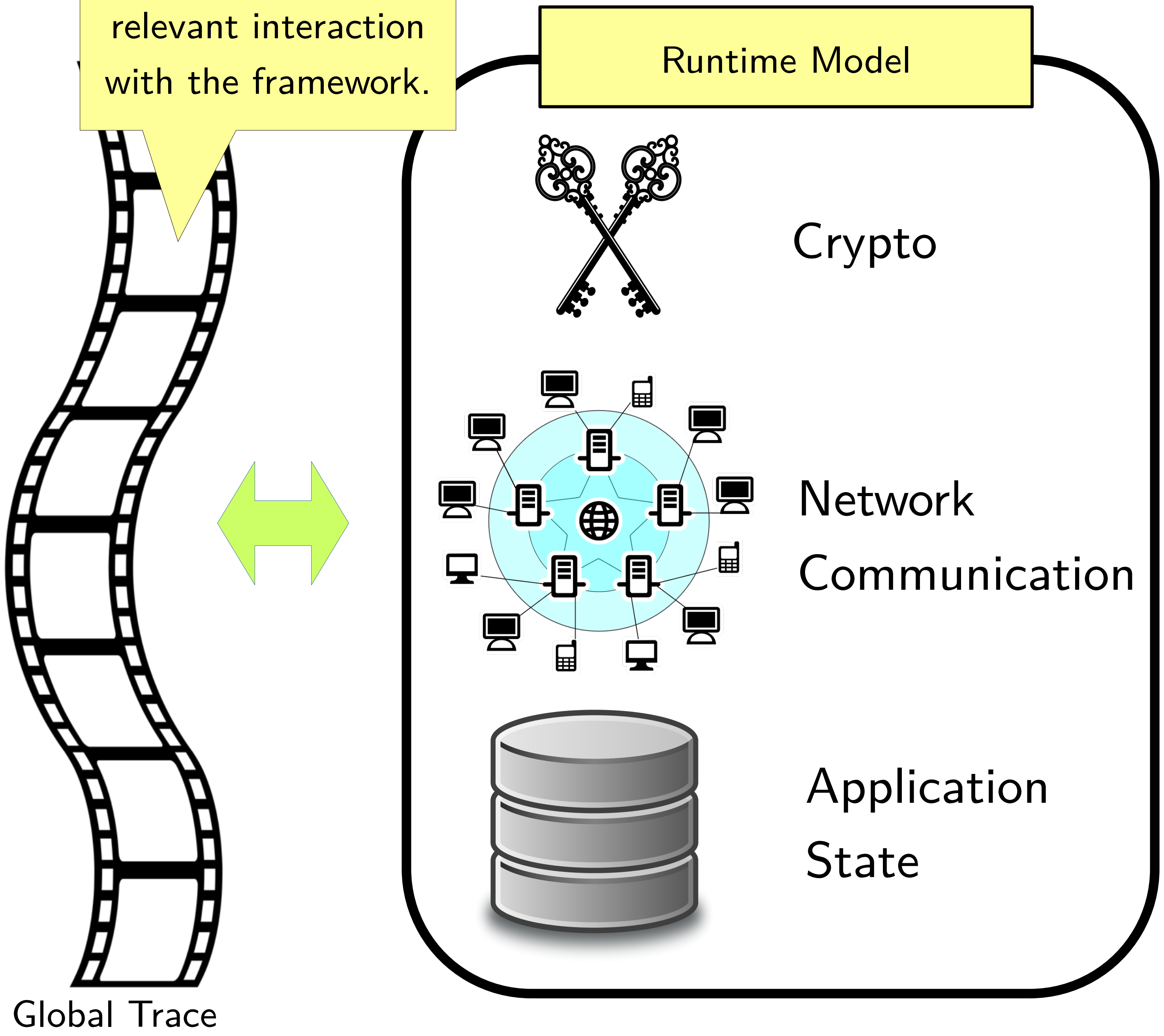
verified

Dolev-Yao* (DY*): Architecture



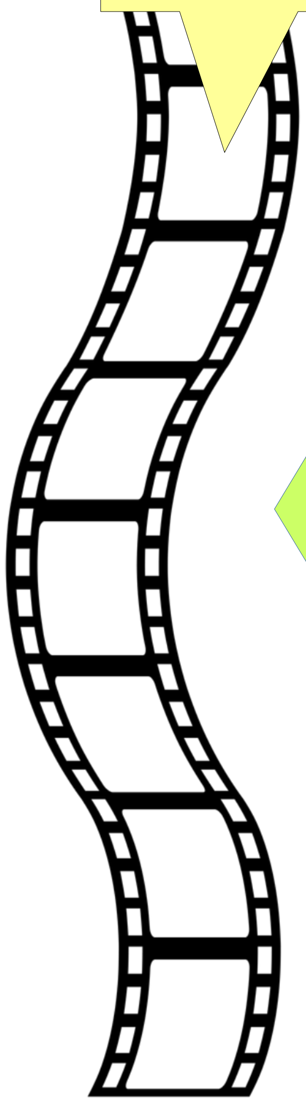
Dolev-Yao* (DY*): Architecture

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



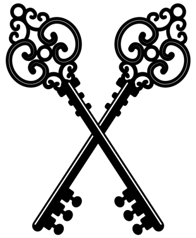
Dolev-Yao* (DY*): Architecture

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

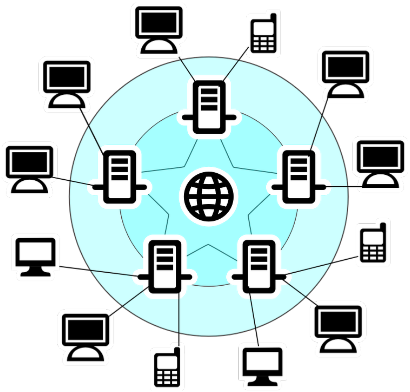


Global Trace

Runtime Model



Crypto

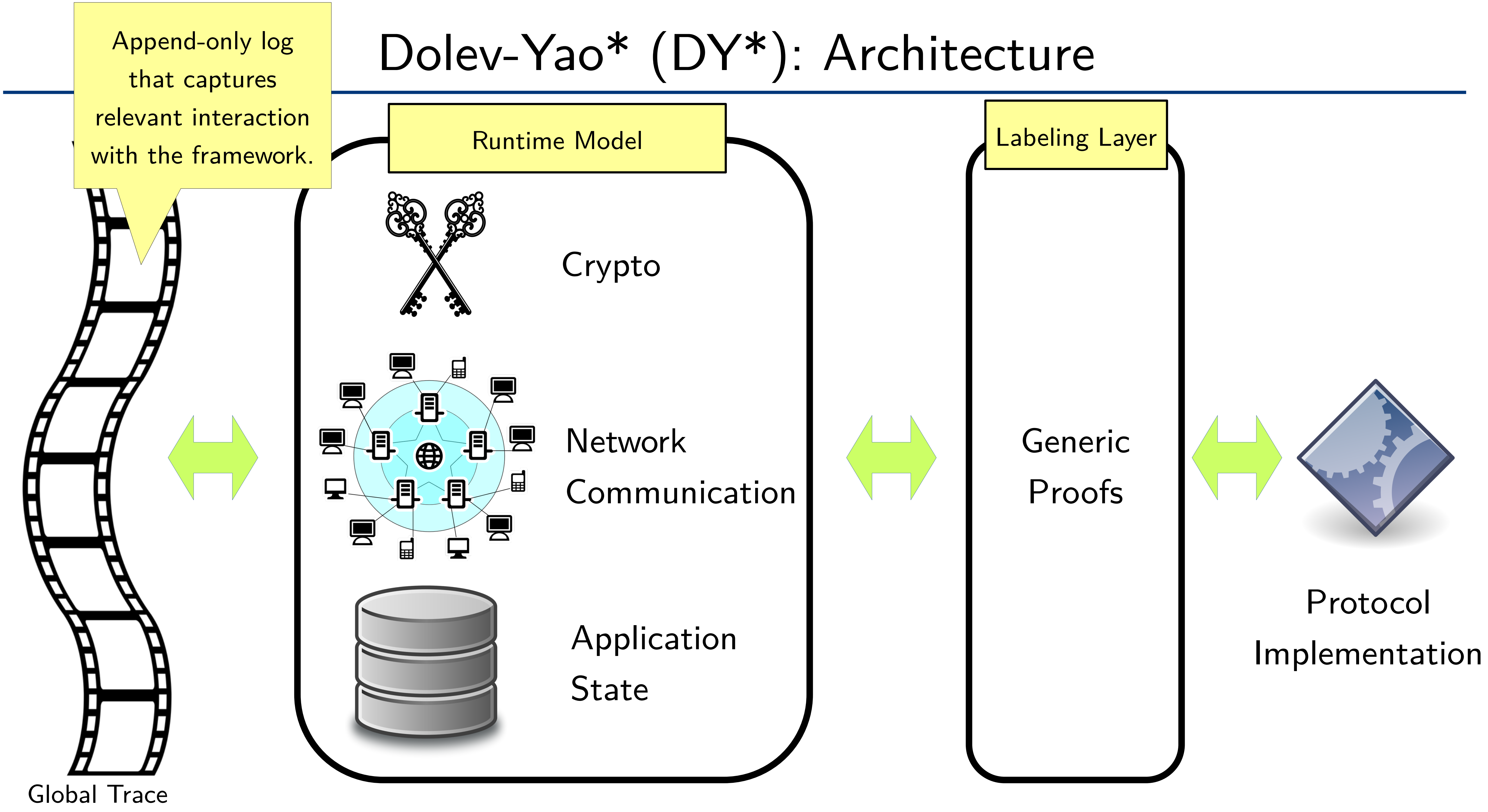


Network Communication

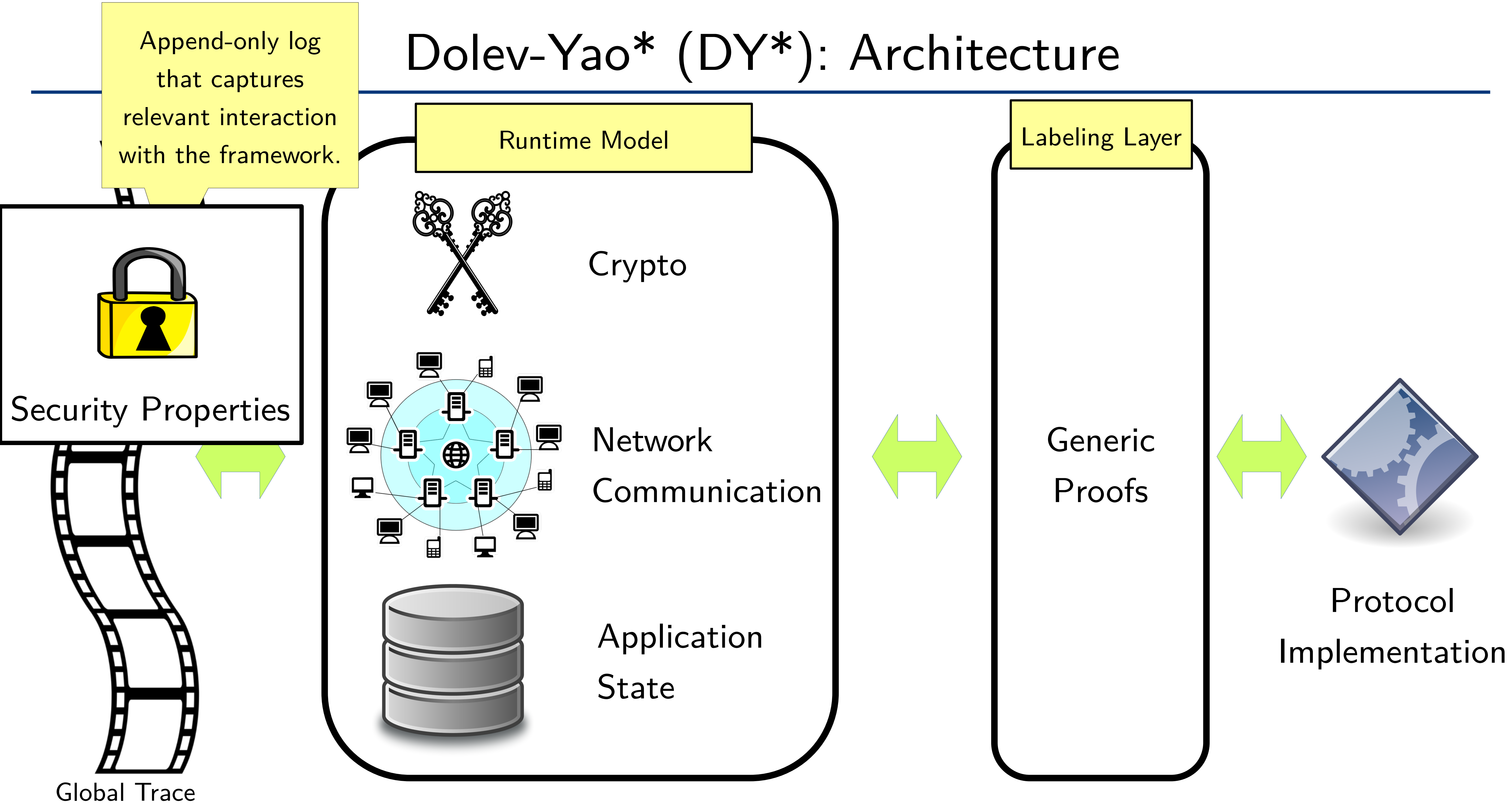
Application State

Protocol Implementation

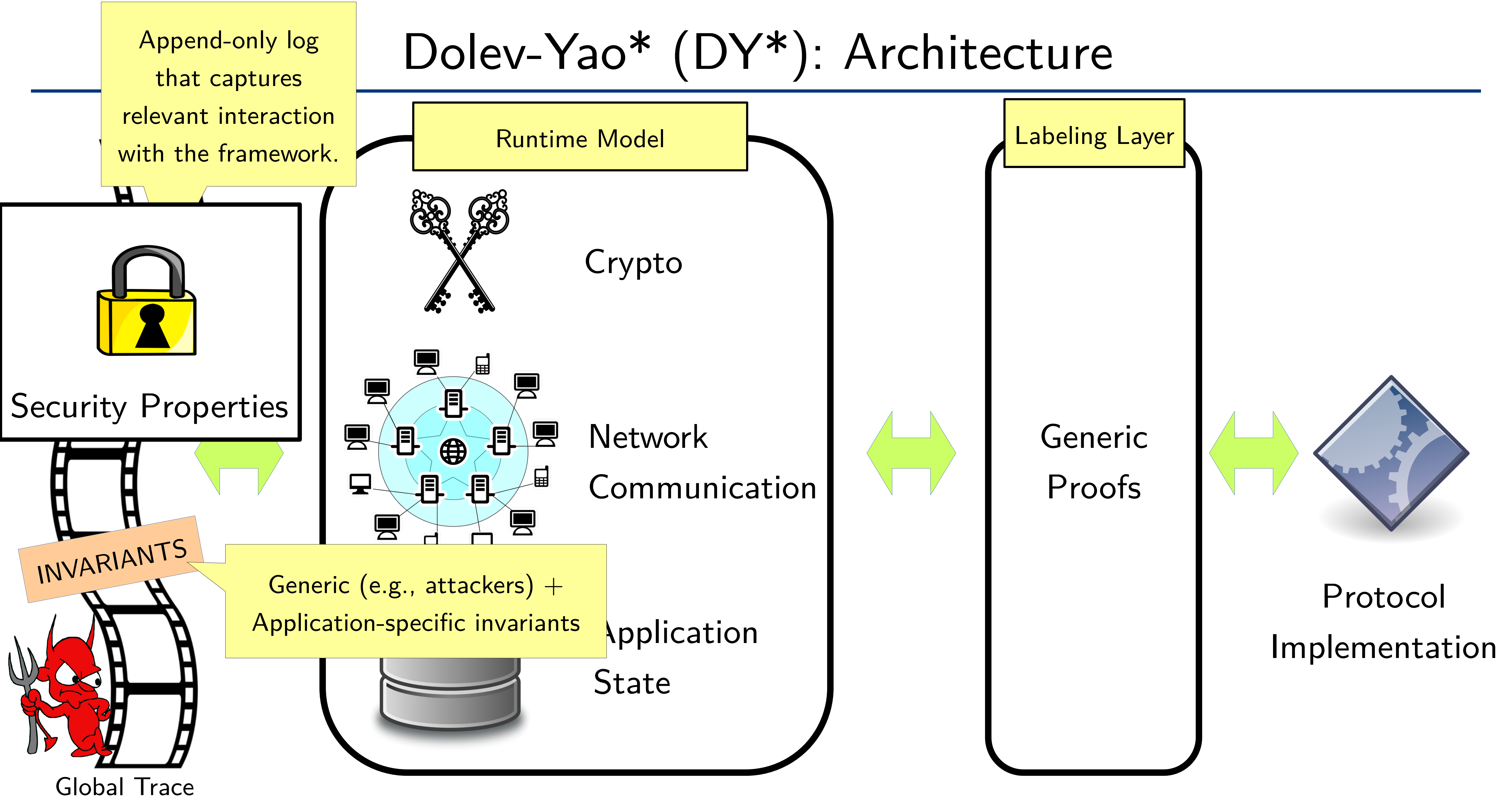
Dolev-Yao* (DY*): Architecture



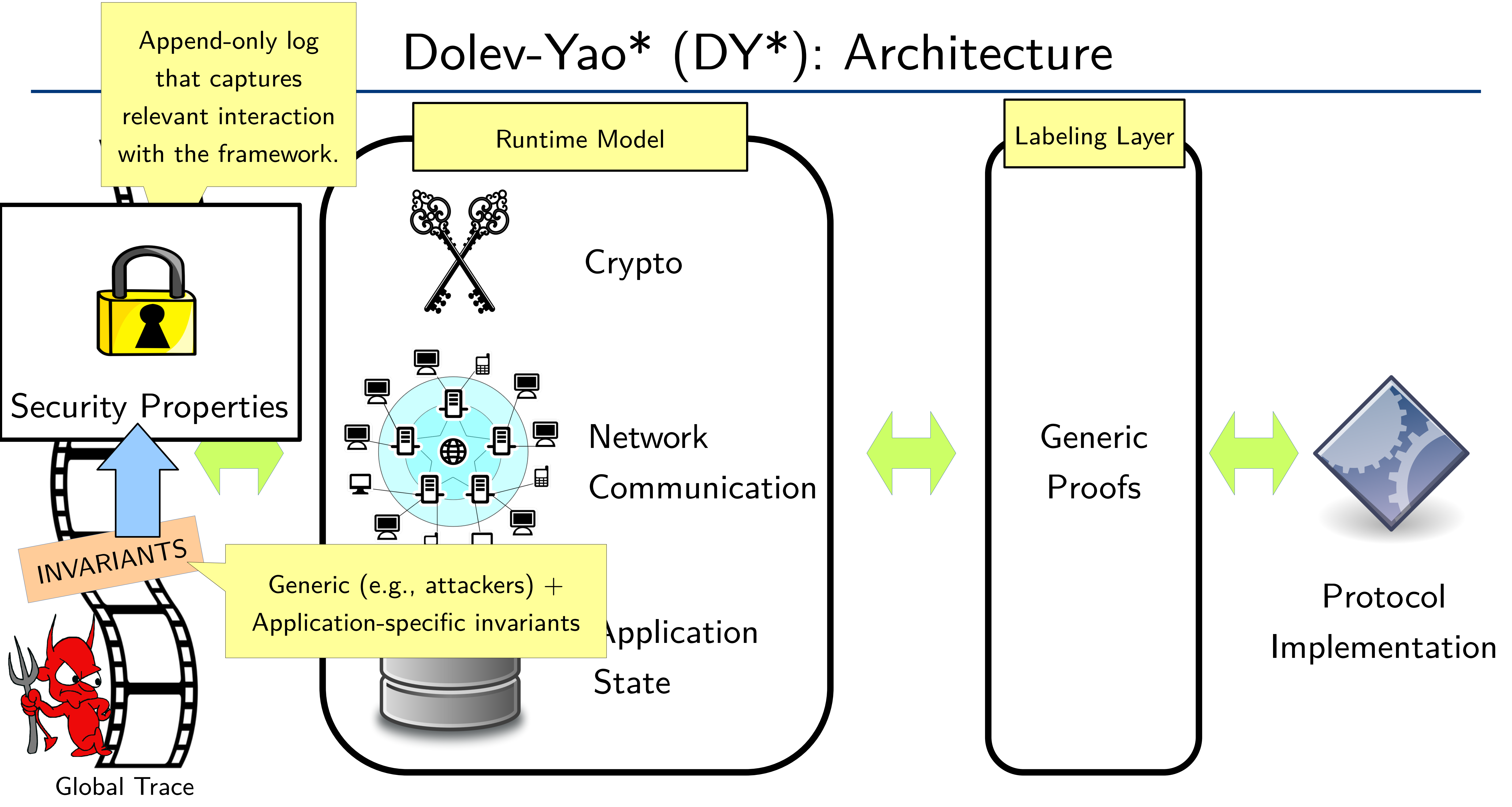
Dolev-Yao* (DY*): Architecture



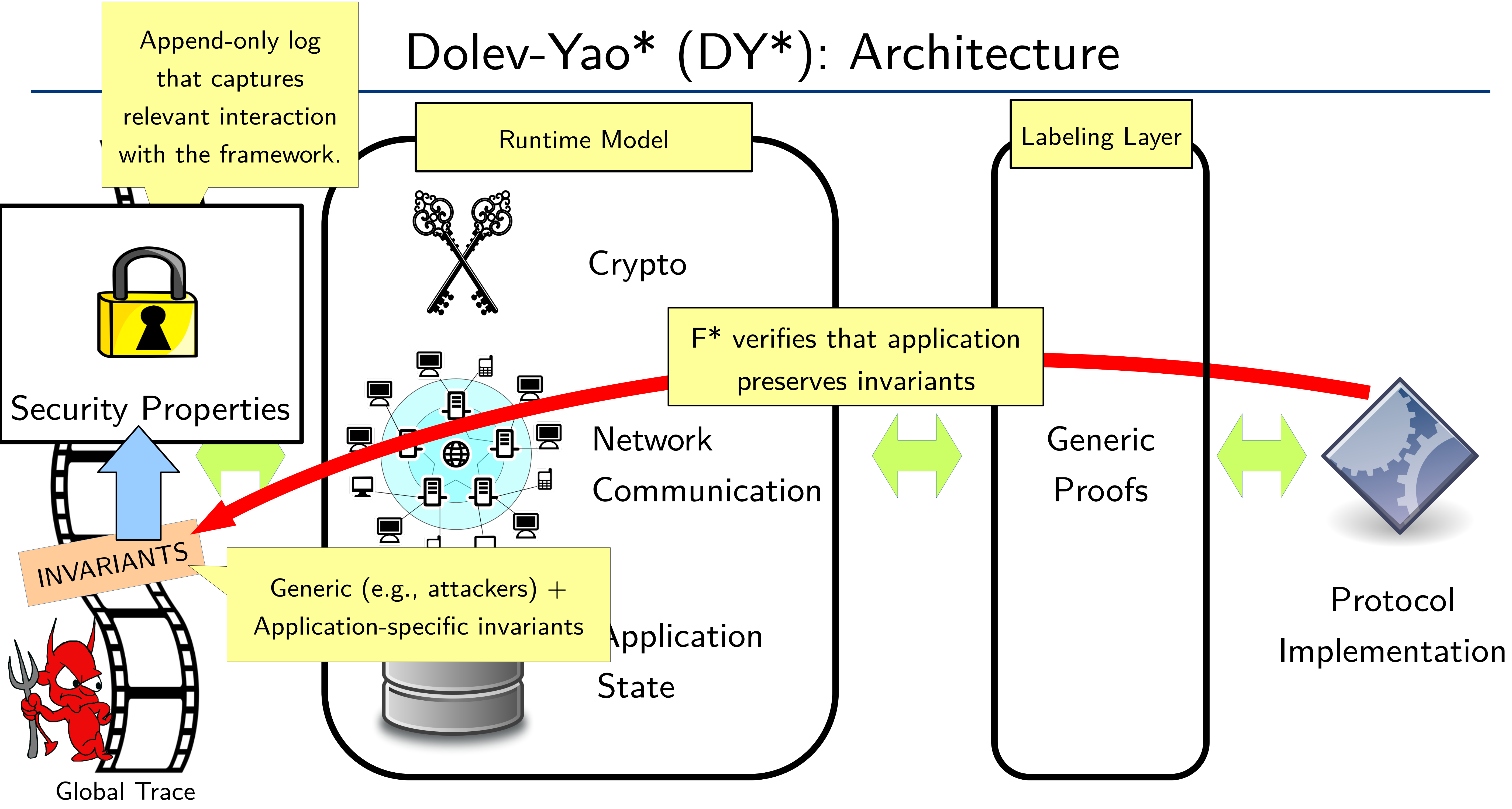
Dolev-Yao* (DY*): Architecture



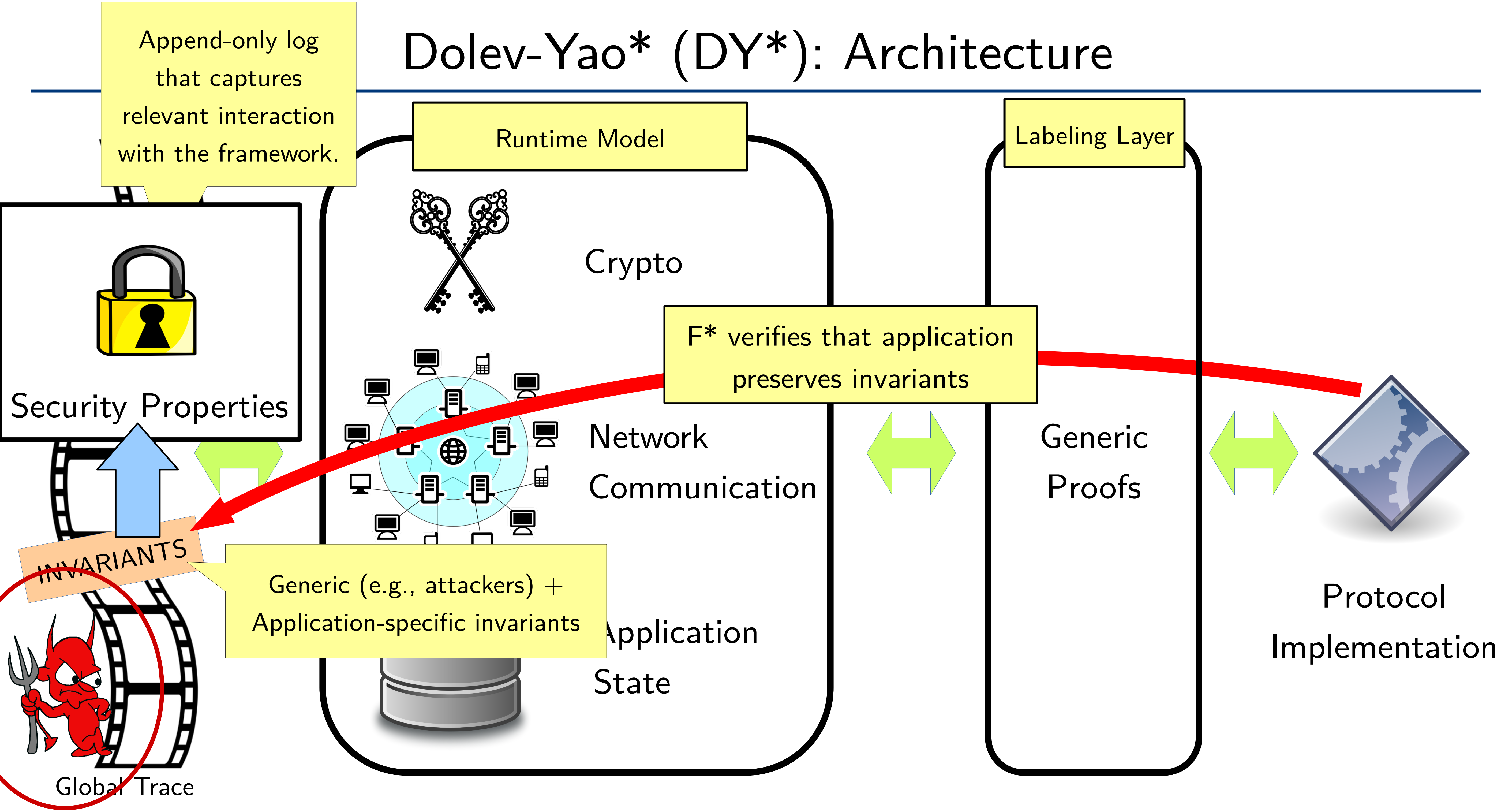
Dolev-Yao* (DY*): Architecture



Dolev-Yao* (DY*): Architecture



Dolev-Yao* (DY*): Architecture

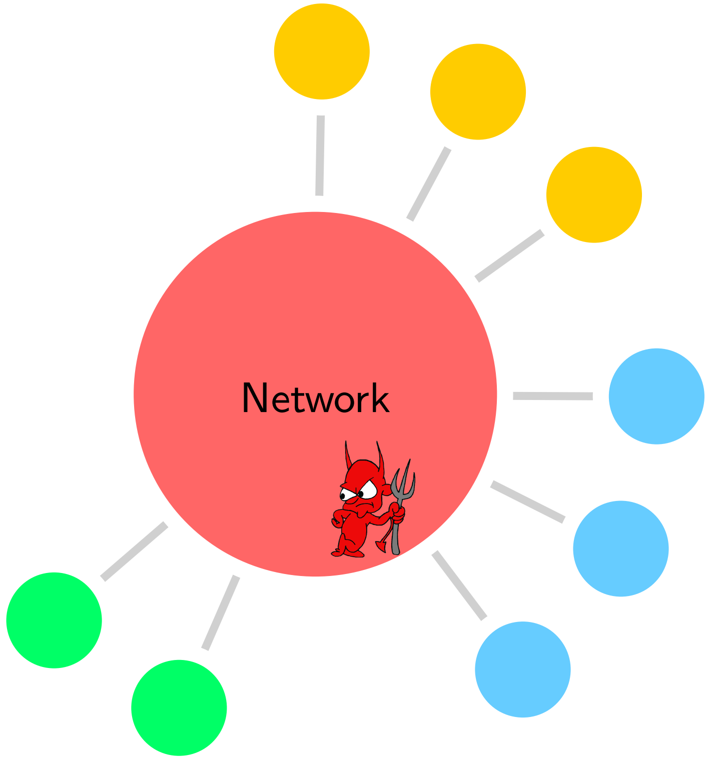


Attacker Model

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 and sessions

Goal: Show that protocol is secure given such an attacker

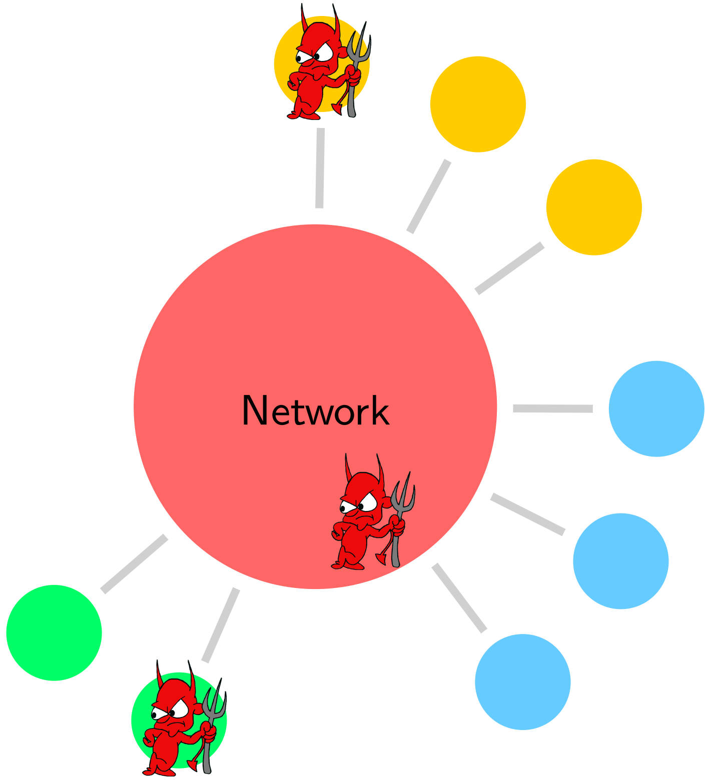


Attacker Model

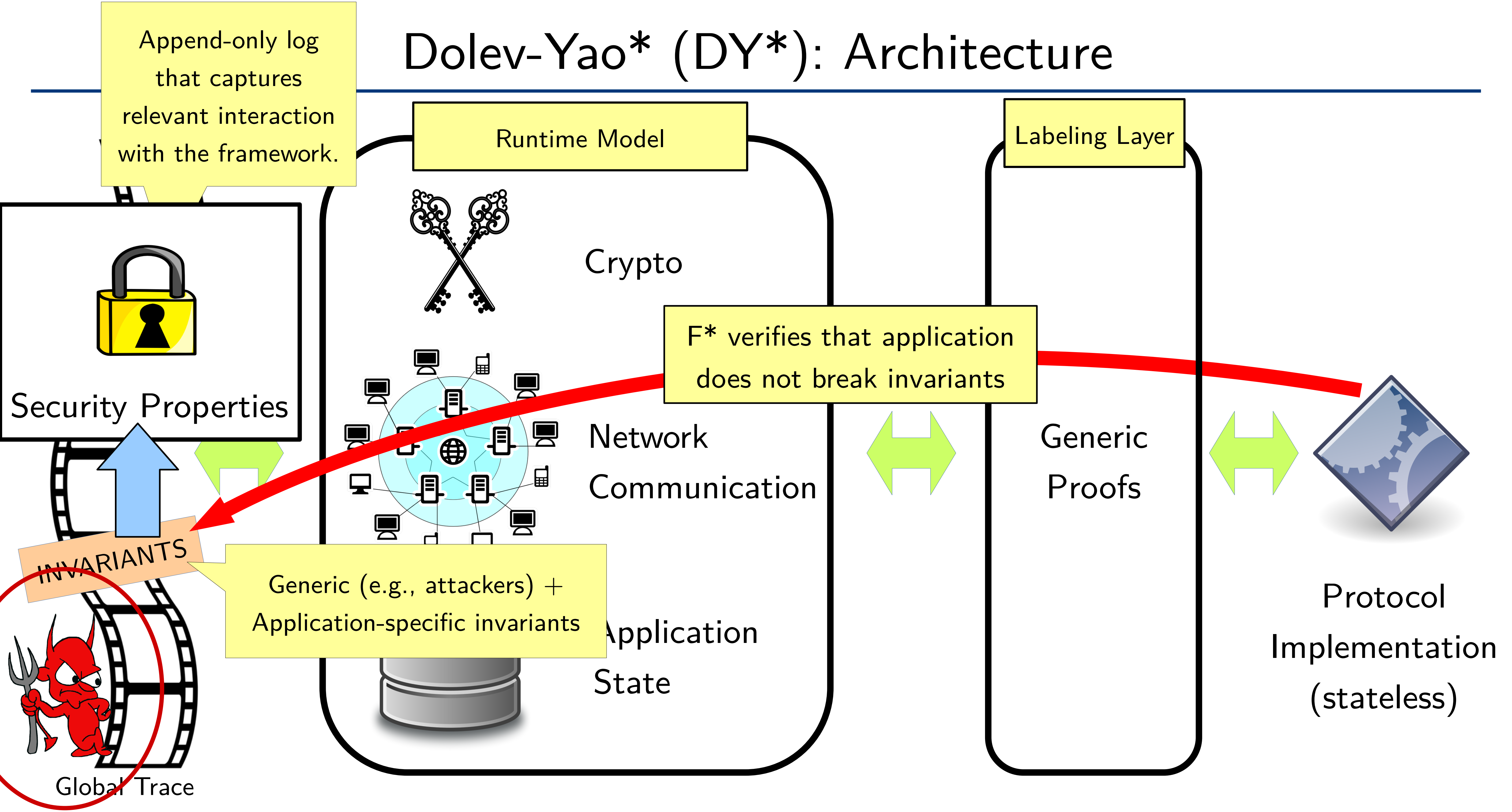
Active Network Attacker

- 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 and sessions

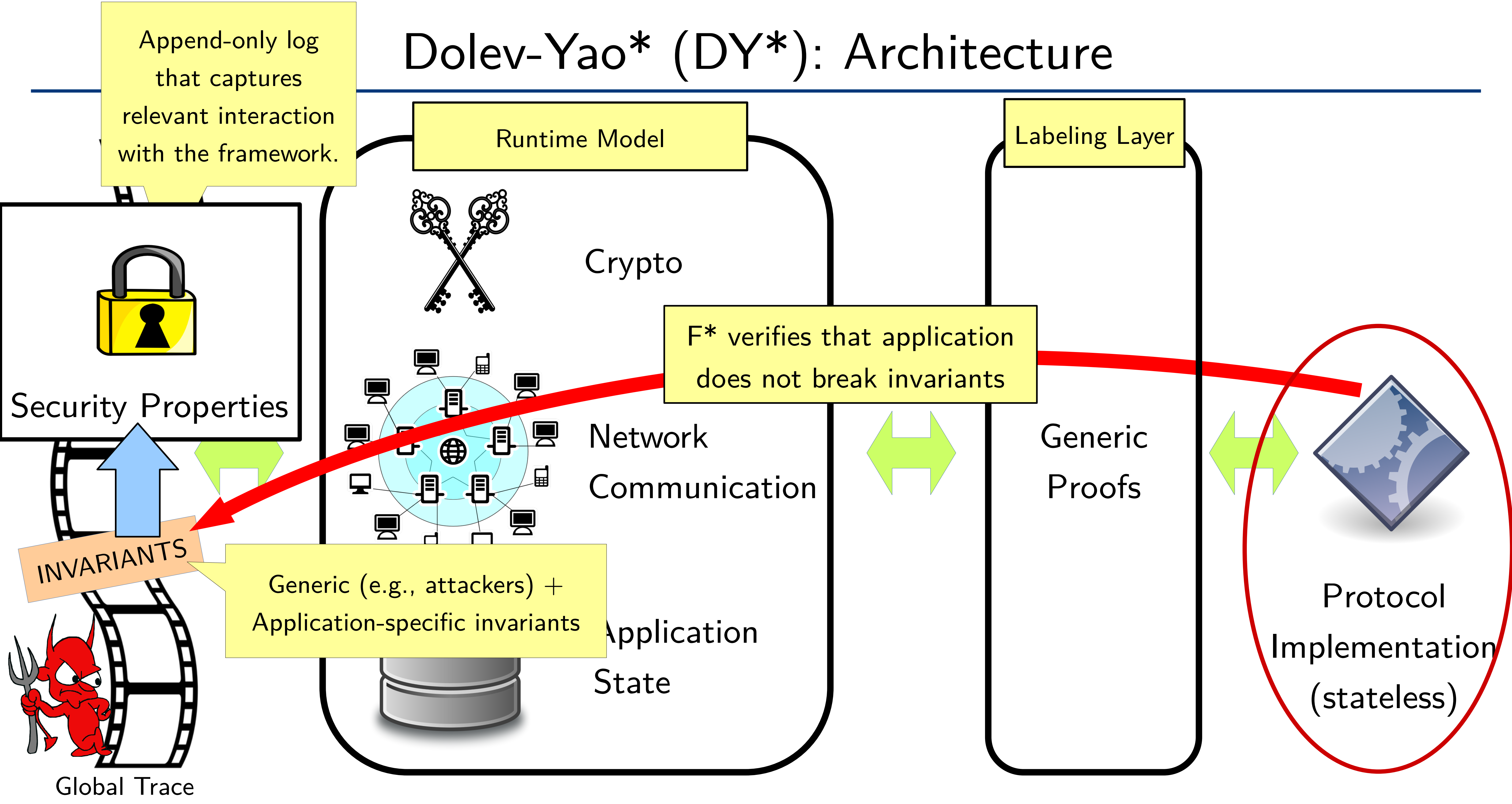
Goal: Show that protocol is secure given such an attacker



Dolev-Yao* (DY*): Architecture



Dolev-Yao* (DY*): Architecture



DY* Example: Protocol Code ISO-DH

```
let initiator_send_msg_1 a b =
  let si = new_session_number a in
  let (|t0,x|) = rand_gen (readers [V a si 0]) (dh_usage "ISODH.dh_key") in
  let gx = dh_pk x in

  let ev = initiate a b gx in
  trigger_event a ev;

  let t1 = global_timestamp () in
  let new_ss_st = InitiatorSentMsg1 b x in
  let new_ss = serialize_valid_session_st t1 a si 0 new_ss_st in
  new_session a si 0 new_ss;

  let t2 = global_timestamp () in
  let msg1 = Msg1 a gx in
  let w_msg1 = serialize_msg t2 msg1 in
  let i = send a b w_msg1 in
  i, si
```

DY* Example: Protocol Code ISO-DH

```
let initiator_send_msg_1 a b =
  let si = new_session_number a in
  let (|t0,x|) = rand_gen (readers [V a si 0]) (dh_usage "ISODH.dh_key") in
  let gx = dh_pk x in

  let ev = initiate a b gx in
  trigger_event a ev;

  let t1 = global_timestamp () in
  let new_ss_st = InitiatorSentMsg1 b x in
  let new_ss = serialize_valid_session_st t1 a si 0 new_ss_st in
  new_session a si 0 new_ss;

  let t2 = global_timestamp () in
  let msg1 = Msg1 a gx in
  let w_msg1 = serialize_msg t2 msg1 in
  let i = send a b w_msg1 in
  i, si
```

Generate a fresh nonce

DY* Example: Protocol Code ISO-DH

```
let initiator_send_msg_1 a b =
  let si = new_session_number a in
  let (|t0,x|) = rand_gen (readers [V a si 0]) (dh_usage "ISODH.dh_key") in
  let gx = dh_pk x in

  let ev = initiate a b gx in
  trigger_event a ev;

  let t1 = global_timestamp () in
  let new_ss_st = InitiatorSentMsg1 b x in
  let new_ss = serialize_valid_session_st t1 a si 0 new_ss_st in
  new_session a si 0 new_ss;

  let t2 = global_timestamp () in
  let msg1 = Msg1 a gx in
  let w_msg1 = serialize_msg t2 msg1 in
  let i = send a b w_msg1 in
  i, si
```

Generate a fresh nonce

Protocol event

DY* Example: Protocol Code ISO-DH

```
let initiator_send_msg_1 a b =
  let si = new_session_number a in
  let (|t0,x|) = rand_gen (readers [V a si 0]) (dh_usage "ISODH.dh_key") in
  let gx = dh_pk x in

  let ev = initiate a b gx in
  trigger_event a ev;

  let t1 = global_timestamp () in
  let new_ss_st = InitiatorSentMsg1 b x in
  let new_ss = serialize_valid_session_st t1 a si 0 new_ss_st in
  new_session a si 0 new_ss;

  let t2 = global_timestamp () in
  let msg1 = Msg1 a gx in
  let w_msg1 = serialize_msg t2 msg1 in
  let i = send a b w_msg1 in
  i, si
```

Generate a fresh nonce

Protocol event

Update state

DY* Example: Protocol Code ISO-DH

```
let initiator_send_msg_1 a b =
  let si = new_session_number a in
  let (|t0,x|) = rand_gen (readers [V a si 0]) (dh_usage "ISODH.dh_key") in
  let gx = dh_pk x in

  let ev = initiate a b gx in
  trigger_event a ev;

  let t1 = global_timestamp () in
  let new_ss_st = InitiatorSentMsg1 b x in
  let new_ss = serialize_valid_session_st t1 a si 0 new_ss_st in
  new_session a si 0 new_ss;

  let t2 = global_timestamp () in
  let msg1 = Msg1 a gx in
  let w_msg1 = serialize_msg t2 msg1 in
  let i = send a b w_msg1 in
  i, si
```

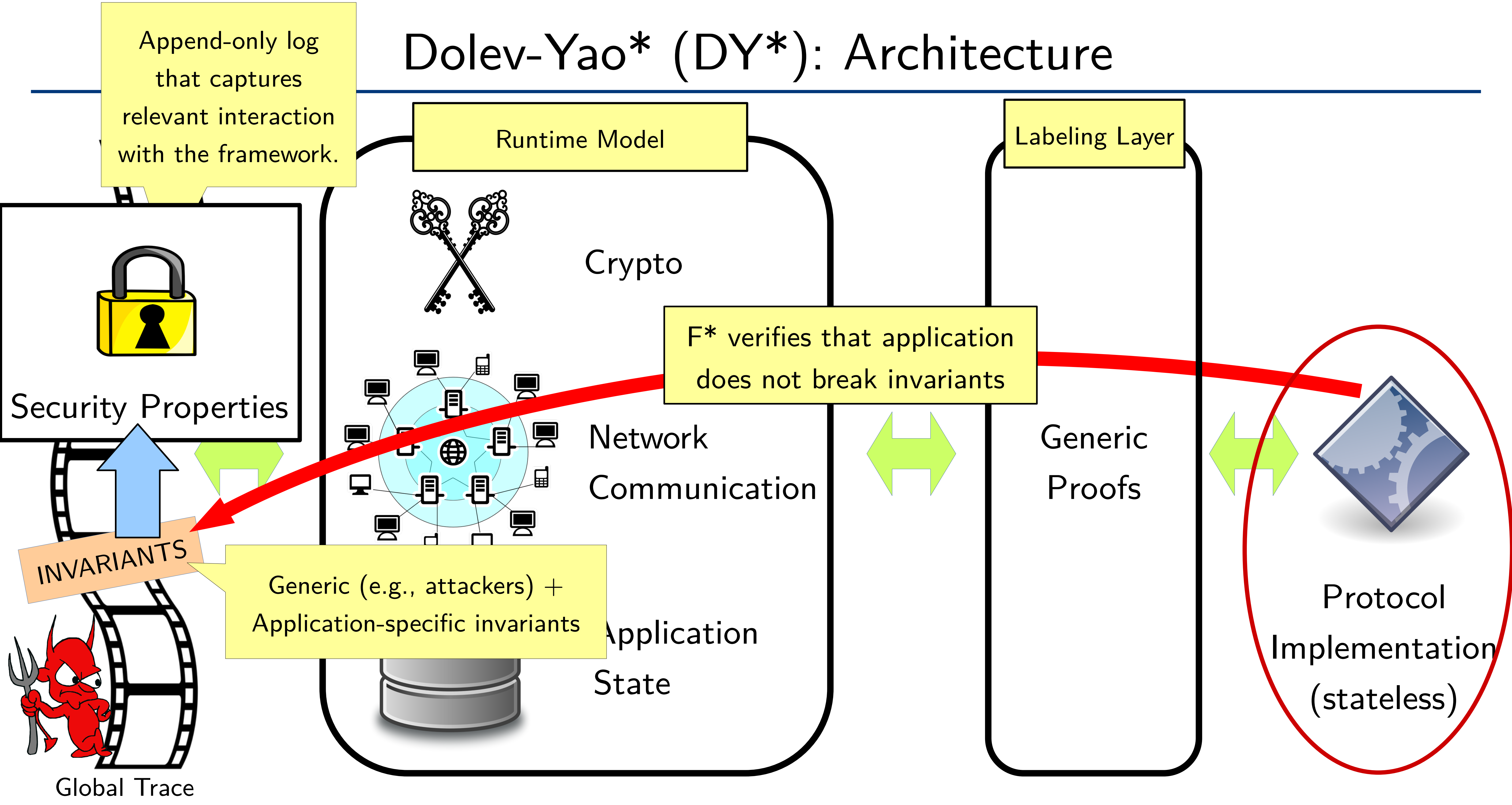
Generate a fresh nonce

Protocol event

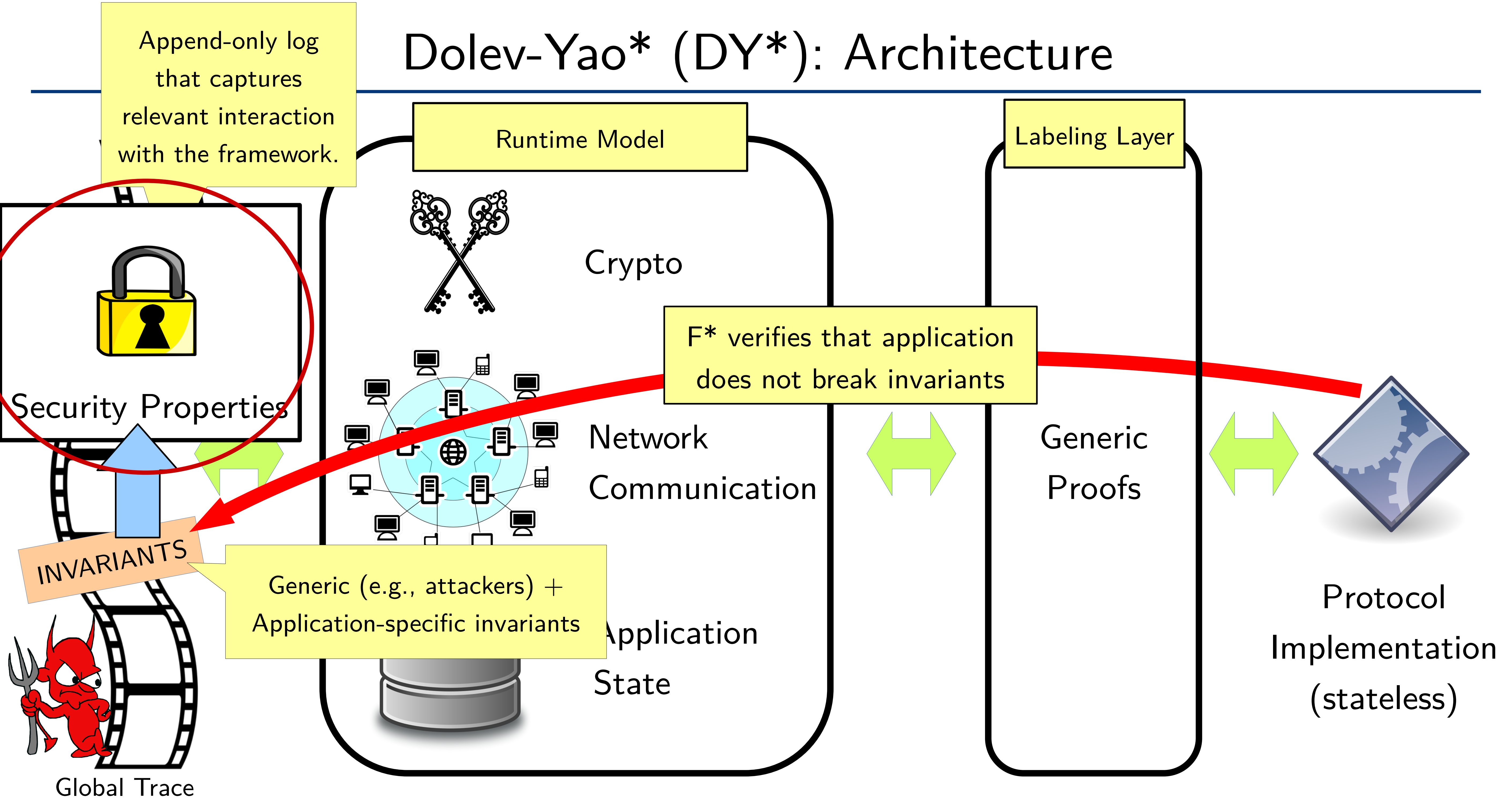
Update state

Send message

Dolev-Yao* (DY*): Architecture



Dolev-Yao* (DY*): Architecture



DY* Example: Security Property

Forward secrecy for an authenticated key exchange protocol (ISO-DH):

```
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)
  ))))
```

DY* Example: Security Property

Forward secrecy for an authenticated key exchange protocol (ISO-DH):

```
val initiator_forward_secrecy_lemma: i:timestamp -> a:principal -> b:pr
    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 ...

DY* Example: Security Property

Forward secrecy for an authenticated key exchange protocol (ISO-DH):

```
val initiator_forward_secrecy_lemma: i:timestamp -> a:principal -> b:pr
    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 ...

DY* Example: Security Property

Forward secrecy for an authenticated key exchange protocol (ISO-DH):

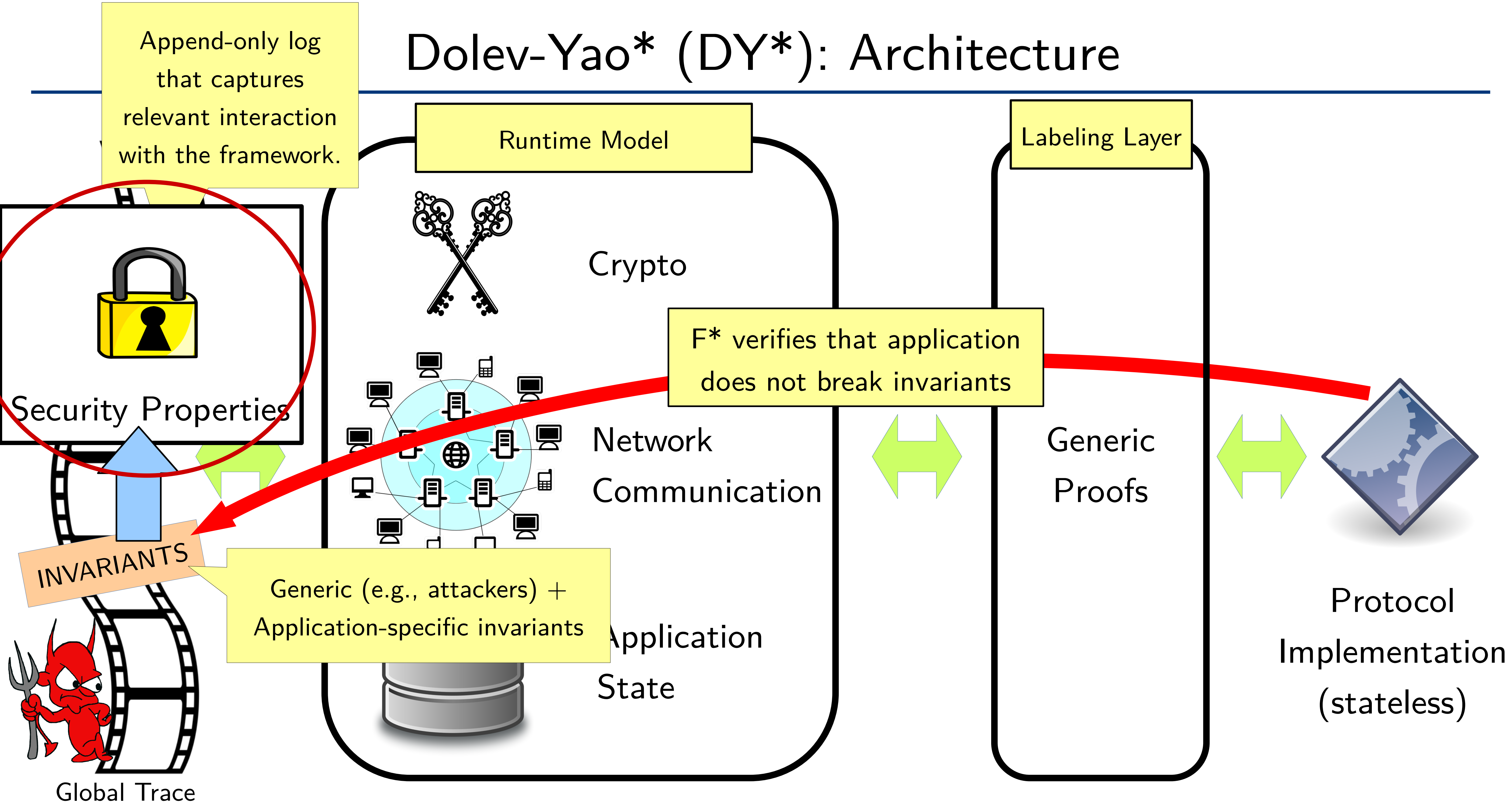
```
val initiator_forward_secrecy_lemma: i:timestamp -> a:principal -> b:pr
    gx:bytes -> gy:bytes -> k:bytes -> LCrypto unit (pki isodh)

  (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 …

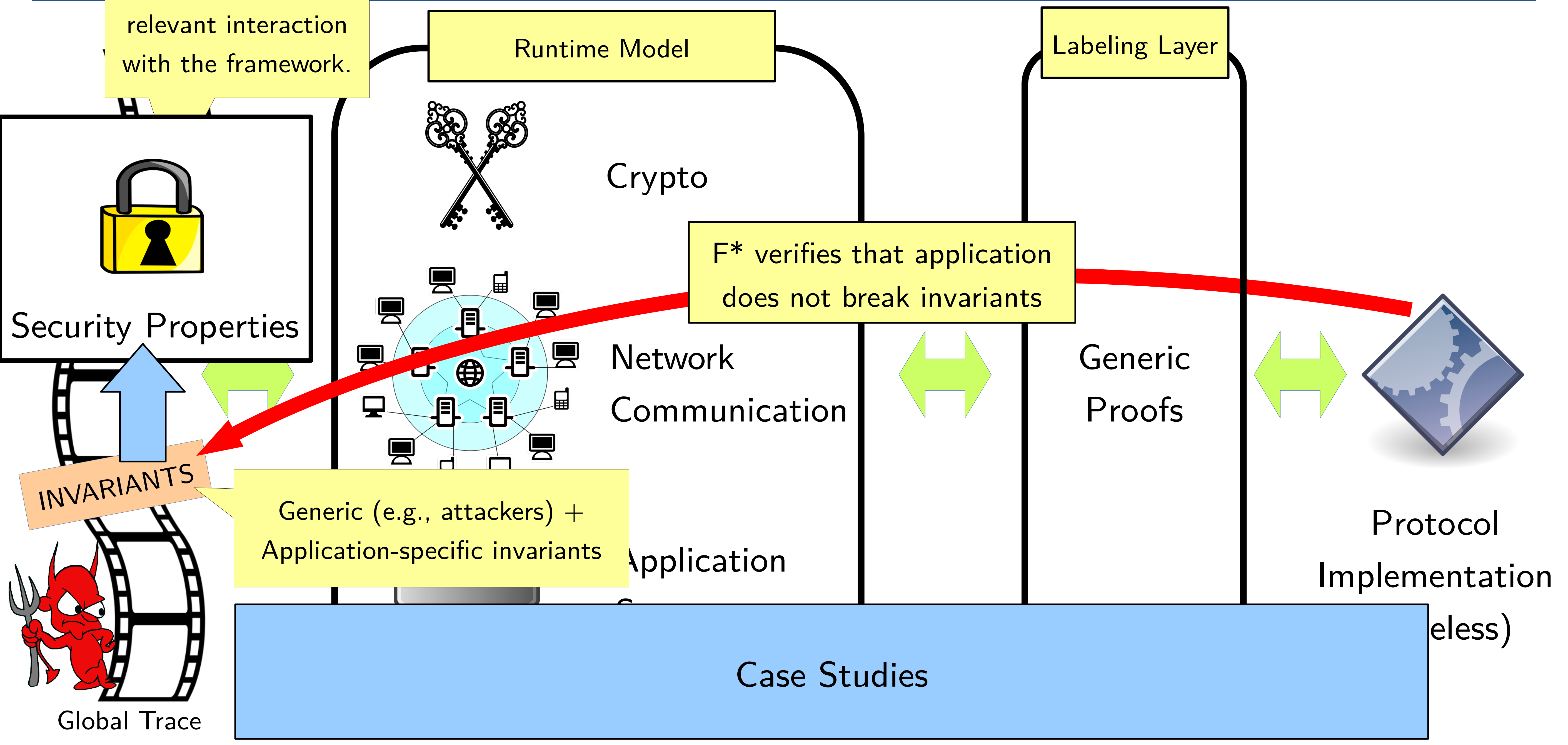
… then Bob was compromised during the protocol run OR …

… the exchanged key k is unknown to the attacker (even if long-term key is later compromised).

Dolev-Yao* (DY*): Architecture



Dolev-Yao* (DY*): Architecture



Case Studies So Far

- Signal Messaging Protocol

Case Studies So Far

- Signal Messaging Protocol
- Automatic Certificate Management Environment (ACME)

Case Studies So Far

- Signal Messaging Protocol
- Automatic Certificate Management Environment (ACME)
- Needham-Schroeder(-Lowe), ISO-DH, and ISO-KEM

Case Studies So Far

- Signal Messaging Protocol

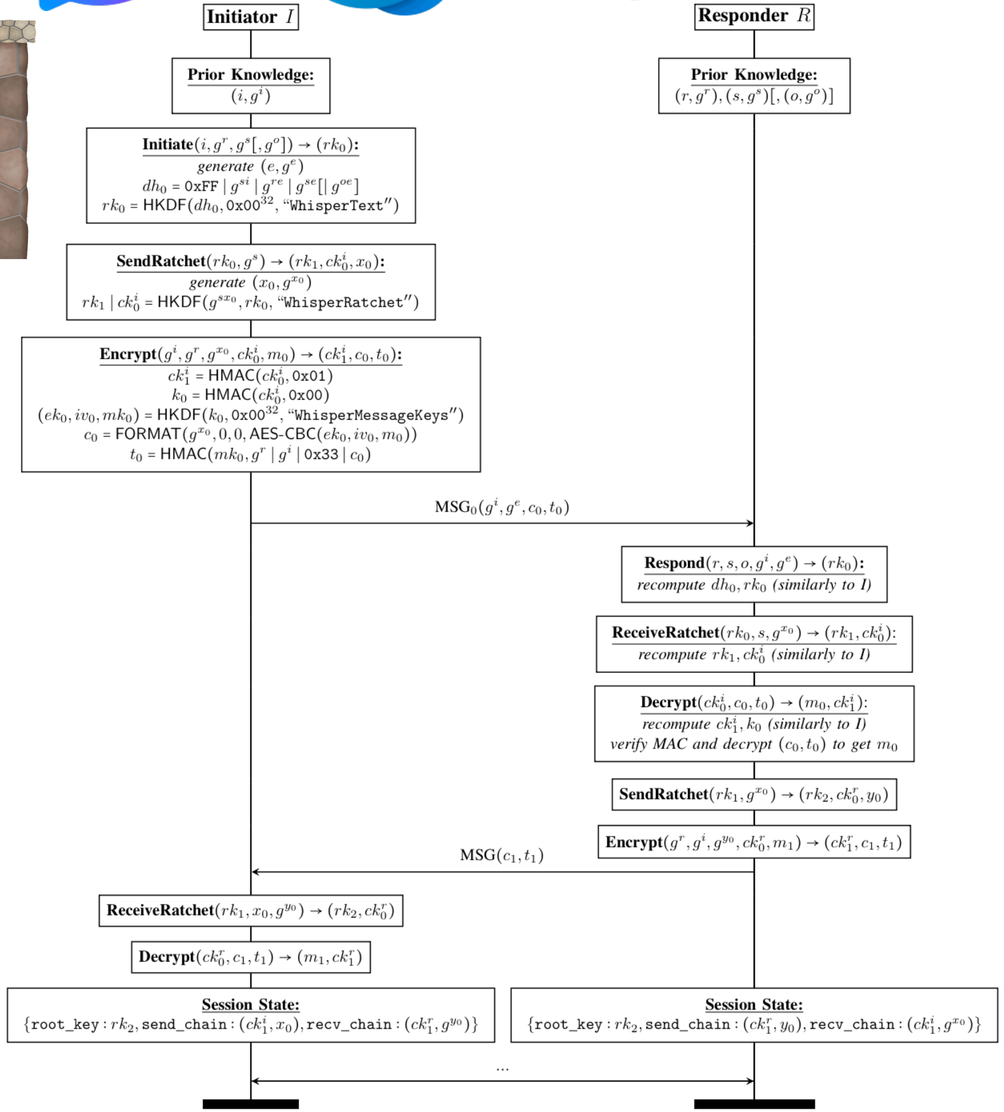
- Automatic Certificate Management Environment (ACME)

- Needham-Schroeder(-Lowe), ISO-DH, and ISO-KEM

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 So Far

- Signal Messaging Protocol
  - Unbounded number of rounds (racheting)
  - Forward Secrecy & Post Compromise Security
- Automatic Certificate Management Environment (ACME)
- Needham-Schroeder(-Lowe), ISO-DH, and ISO-KEM

Case Studies So Far

- Signal Messaging Protocol
  - Unbounded number of rounds (racheting)
  - Forward Secrecy & Post Compromise Security
- Automatic Certificate Management Environment (ACME)
- Needham-Schroeder(-Lowe), ISO-DH, and ISO-KEM

Case Studies So Far

- Signal Messaging Protocol
  - Unbounded number of rounds (racheting)
  - Forward Secrecy & Post Compromise Security
- Automatic Certificate Management Environment (ACME)
- Needham-Schroeder(-Lowe), ISO-DH, and ISO-KEM

Automatic Certificate Management Environment (ACME)

- Main User: Let's Encrypt
  - Over 250M active domains with LE certificates
  - Currently issuing ∼ 2M certificates/day
  - Issued more than 1B certificates so far
- ACME is supported by many other certification authorities (e.g., Buypass, DigiCert, GlobalSign, ZeroSSL)

Statistics from `https://letsencrypt.org/stats/`

Automatic Certificate Management Environment (ACME)

ACME is extremely security critical

- Two previous formal analyses (ProVerif, Tamarin)
  - Very abstract models
  - Only consider certificates for a single domain
  - No loops of subprotocols
  - No precise state management
  - No details of message formats

Limited by the used tools – DY* analysis overcomes these limitations

- Our analysis is one of the largest & most in-depth formal security analyses in the literature (16,000 LoC)
- ACME client model can interoperate with real-world server

Case Studies So Far

- Signal Messaging Protocol
  - Unbounded number of rounds (racheting)
  - Forward Secrecy & Post Compromise Security
- Automatic Certificate Management Environment (ACME)
  - One of the largest & most in-depth formal security analyses in the literature (16,000 LoC)
  - ACME client model can interoperate with real-world server
- Needham-Schroeder(-Lowe), ISO-DH, and ISO-KEM

Case Studies So Far

- Signal Messaging Protocol
  - Unbounded number of rounds (racheting)
  - Forward Secrecy & Post Compromise Security
- Automatic Certificate Management Environment (ACME)
  - One of the largest & most in-depth formal security analyses in the literature (16,000 LoC)
  - ACME client model can interoperate with real-world server
- 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

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!
  - WIM*: mechanize the **Web Infrastructure Model**
  - Concrete DY*: fully verified implementations
  - Equivalence properties
  - Computational analysis

Conclusion & Future Work

See [S&P ‘14, ESORICS ‘15, CCS ‘15, CCS ‘16, CSF ‘17, S&P ‘19]

- 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!
  - WIM*: mechanize the **Web Infrastructure Model**
  - Concrete DY*: fully verified implementations
  - Equivalence properties
  - Computational analysis

Conclusion & Future Work

See [S&P ‘14, ESORICS ‘15, CCS ‘15, CCS ‘16, CSF ‘17, S&P ‘19]

- 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!
  - WIM*: mechanize the **Web Infrastructure Model**
  - Concrete DY*: fully verified implementations
  - Equivalence properties
  - Computational analysis

We are hiring!

Postdoc position in Stuttgart

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

See [S&P '14, ESORICS '15, CCS '15, CCS '16, CSF '17, S&P '19]

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

We are hiring!
Postdoc position in Stuttgart

Find more information on: **https://sec.uni-stuttgart.de**

Conclusion & Future Work

See [S&P '14, ESORICS '15, CCS '15, CCS '16, CSF '17, S&P '19]

- 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!
  - WIM*: mechanize the **Web Infrastructure Model**
  - Concrete DY*: fully verified implementations
  - Equivalence properties
  - Computational analysis

We are hiring!
Postdoc position in Stuttgart

Find more information on: **https://sec.uni-stuttgart.de**

Thank you!