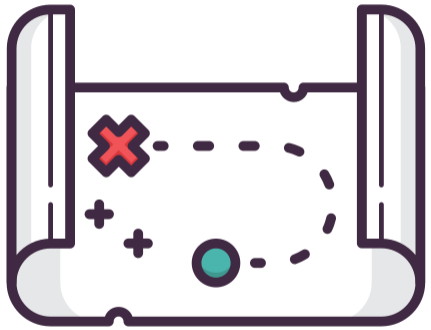


Noise Explorer

Fully automated modeling, analysis and verification
for arbitrary Noise protocols

IACR Real World Crypto
Symposium 2019
San Jose, California

Nadim Kobeissi
Karthikeyan Bhargavan
PROSECCO

Noise Protocol Framework: What is it?

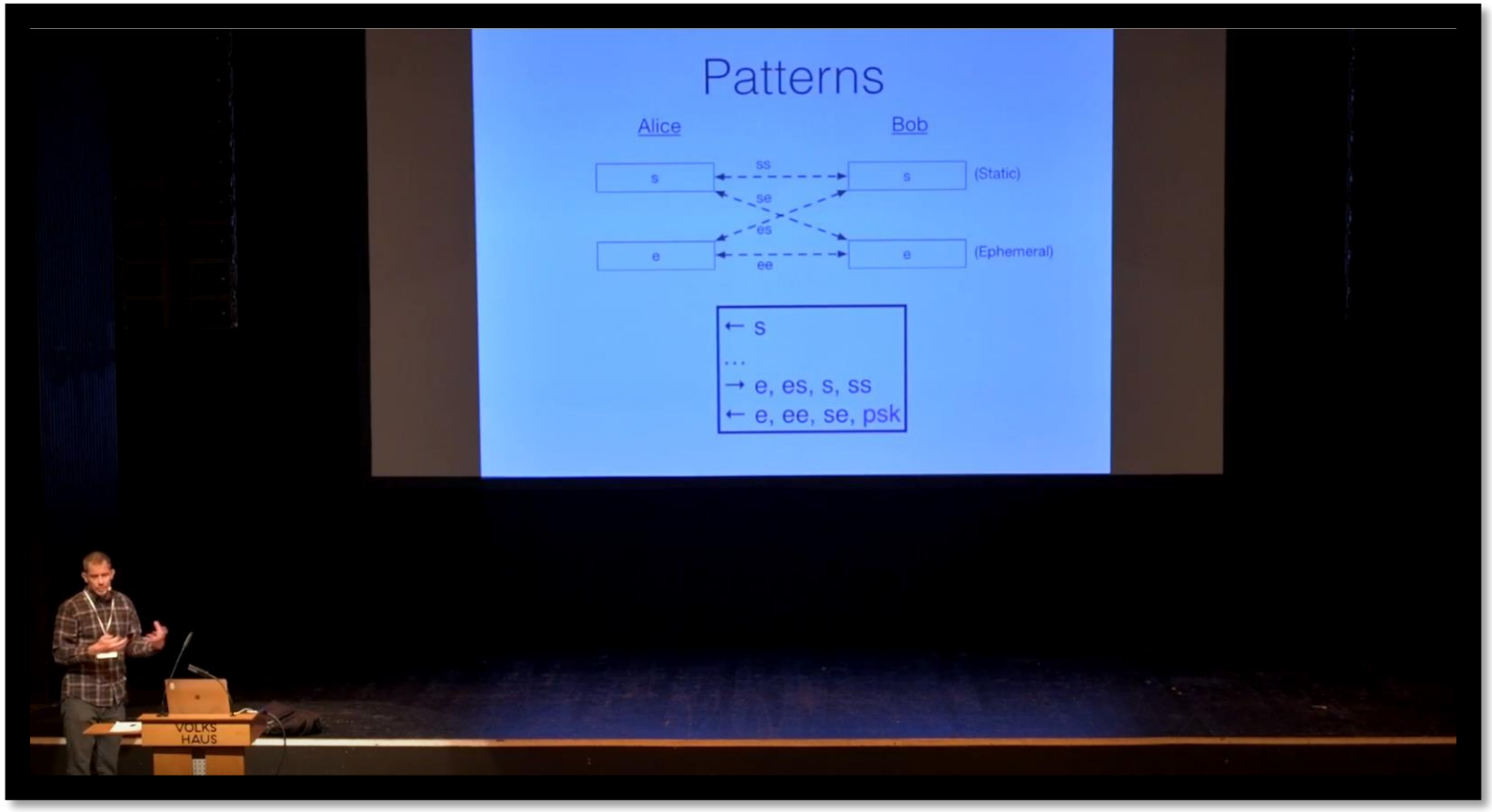
A Framework for Secure Channel Protocols

- Based on Diffie-Hellman key agreement.
- Simple language for describing messages.
- From message description, complex state transformations are derived.
- Author: Trevor Perrin.

Example Noise Handshake Pattern

```
NK:
  <- s
  ...
  -> e, es
  <- e, ee
```

Trevor Perrin's Talk at RWC2018



https://youtu.be/3gipxdJ22iM

Understanding the Notation

Handshake Pattern Notation

- `s, e`: local static and ephemeral key pairs. Automatically generated when they appear in a message.
- `ss, se, es, ee`: Diffie-Hellman operations. Automatically mixed into state.
- Once we have shared secret agreement, encryption on certain payload elements kicks in automatically.

Example Noise Handshake Pattern

```
IK:
  <- s
  ...
  -> e, es, s, ss
  <- e, ee, se
```

Handshake State Machine

State Transformation Functions

- Defined cryptographic operations: `EncryptAndHash, HKDF`, etc.
- Defined local state objects: `CipherState, SymmetricState, HandshakeState`.
- Defined state transformations when processing tokens in messages: `MixHash, MixKey`, etc.

Example Noise Handshake Pattern

```
XX:
  -> e
  <- e, ee, s, es
  -> s, se
```

Popular Adaptations of Noise

WhatsApp

```
XX:
  -> e
  <- e, ee, s, es
  -> s, se

IK:
  <- s
  ...
  -> e, es, s, ss
  <- e, ee, se
```

WireGuard

```
IKpsk2:
  <- s
  ...
  -> e, es, s, ss
  <- e, ee, se, psk
```

Security Goals in the Noise Specification

Grade Based System

- Authentication: three grades:
  - 0, 1, 2
- Confidentiality: six grades:
  - 0, 1, 2, 3, 4, 5
- Identity hiding:
  - Not currently evaluated by Noise Explorer.

Example Noise Handshake Pattern

```
KN:
  -> s
  ...
  -> e                    0    0
  <- e, ee, se            0    3
  ->                      2    1
  <-                      0    5
```

Security Goals in the Noise Specification

Authentication Grades

- Authentication **0**: No authentication.
  - *"This payload may have been sent by any party, including an active attacker."*
- Authentication **1**: Sender authentication vulnerable to KCI.
  - *"If the recipient's long-term private key has been compromised, this authentication can be forged."*
- Authentication **2**: Sender authentication resistant to KCI.
  - *"Assuming the corresponding private keys are secure, this authentication cannot be forged."*

Example Noise Handshake Pattern

```
KN:
  -> s
  ...
  -> e                    0    0
  <- e, ee, se            0    3
  ->                      2    1
  <-                      0    5
```

Security Goals in the Noise Specification

Confidentiality Grades

- Confidentiality **0**: No confidentiality.
  - *"This payload is sent in cleartext."*
- Confidentiality **1**: Encryption to ephemeral recipient.
  - *"This payload has forward secrecy, since encryption involves an ephemeral-ephemeral DH ("ee"). However, the sender has not authenticated the recipient, so this payload might be sent to any party, including an active attacker."*
- Confidentiality **2**: Forward secrecy for sender compromise only, vulnerable to replay.
  - *"If the recipient's static private key is compromised, even at a later date, this payload can be decrypted. This message can also be replayed, since there's no ephemeral contribution from the recipient."*

Example Noise Handshake Pattern

```
KN:
   -> s

   ...
   -> e                    0    0
   <- e, ee, se            0    3
   ->                      2    1
   <-                      0    5
```

Security Goals in the Noise Specification

Confidentiality Grades

- Confidentiality **3**: Weak forward secrecy.
  - *"The recipient's alleged ephemeral public key may have been forged by an active attacker. In this case, the attacker could later compromise the recipient's static private key to decrypt the payload."*
- Confidentiality **4**: Weak forward secrecy if sender's private key was compromised.
  - *"If the sender's static private key was previously compromised, the recipient's alleged ephemeral public key may have been forged by an active attacker. In this case, the attacker could later compromise the intended recipient's static private key to decrypt the payload."*
- Confidentiality **5**: Strong forward secrecy.
  - *"Assuming the ephemeral private keys are secure, and the recipient is not being actively impersonated by an attacker that has stolen its static private key, this payload cannot be decrypted."*

Example Noise Handshake Pattern

```
KN:
  -> s
  ...
  -> e                    0    0
  <- e, ee, se            0    3
  ->                      2    1
  <-                      0    5
```

So Many Security Goals!

50+ Handshake Patterns in the Spec Alone

- How do we verify all of these protocols against $(50+ \cdot 10) = 500+$ security queries?

Noise Allows for Use-Case Specific Protocols

- TLS isn't (and shouldn't be) the answer to everything.
- How can we ascertain which security promises *any* Noise Handshake Pattern can give?

Noise Explorer: Design and Formally Verify any Noise Handshake Pattern

- **Design Noise Protocols**: Immediate to-spec validity checks, helpful visualizations.
- **Generate Models for Formal Verification**: Symbolic models for ProVerif.
  - Top-level processes.
  - Sophisticated queries for all security goals.
  - Compromised principal (Charlie).
- **Noise Explorer Compendium**: Formal verification results for 50+ Noise Handshake Patterns.
- **NEW: Generate Implementations**: Generates full implementations of your Noise Handshake Pattern in JS and Go.

What is Formal Verification with ProVerif?

Automated formal verification…

- Beating the *“code first, specify later”* (if ever) methodology.
- **Two main models:** Symbolic model and computational model.
- We use the symbolic model, where we can model protocol flows and try to find contradictions to security queries.

…with ProVerif.

- Developed at INRIA Paris by Bruno Blanchet and team.
- Check it out: http://prosecco.gforge.inria.fr/personal/bblanche/proverif/
- I defended my Ph.D. thesis last month, which has many, many, *many* uses of ProVerif: https://hal.inria.fr/tel-01950884

Generating Applied Pi Models for ProVerif

Components to Model

- In ProVerif, all cryptographic primitives are perfect symbolic black-boxes with no algebraic properties.

Diffie-Hellman in ProVerif

```
fun dhexp(key, key):key.
equation forall a:key, b:key;
        dhexp(b, dhexp(a, g)) =
dhexp(a, dhexp(b, g)).
```

Generating Applied Pi Models for ProVerif

Components to Model

- In ProVerif, all cryptographic primitives are perfect symbolic black-boxes with no algebraic properties.
- Encryption is a PRP, hashing is a PRF, etc.

AEAD in ProVerif

```
fun encrypt(key, nonce, bitstring,
bitstring):bitstring.


fun decrypt(key, nonce, bitstring,
bitstring):aead reduc
    forall k:key, n:nonce, ad:bitstring,
    plaintext:bitstring;


    decrypt(k, n, ad, encrypt(k, n, ad,
    plaintext)) = aeadpack(true, ad,
    plaintext).
```

Generating Applied Pi Models for ProVerif

Components to Model

- In ProVerif, all cryptographic primitives are perfect symbolic black-boxes with no algebraic properties.
- Encryption is a PRP, hashing is a PRF, etc.
- Common state management library for all generated models.

State Management in ProVerif

```
letfun mixKeyAndHash(ss:symmetricstate,
input_key_material:key) =
        let (cs:cipherstate, ck:key,
h:bitstring) = symmetricstateunpack(ss) in
        let (ck:key, temp_h:key,
temp_k:key) = hkdf(ck, input_key_material)
in
        let (cs:cipherstate, temp_ck:key,
h:bitstring) =
symmetricstateunpack(mixHash(symmetricstat
epack(cs, ck, h), key2bit(temp_h))) in
        symmetricstatepack(initializeKey(t
emp_k), ck, h).
```

Our Findings

Pattern	Auth.	Conf.	Pattern	Auth.	Conf.	Pattern	Auth.	Conf.
N	0	2	X1N	0 0 0 0 2	0 1 1 3 1	I1K1	0 4 4 4 4	0 1 5 5 5
K	1	2	X1K	0 2 0 4 4 4	2 1 5 3 5 5	I1X	0 4 4 4 4	0 1 5 5 5
X	1	2	XK1	0 2 4 4 4	0 1 5 5 5	IX1	0 0 4 4 4	0 3 3 5 5
NN	0 0 0	0 1 1	X1K1	0 2 0 4 4 4	0 1 5 3 5 5	I1X1	0 0 4 4 4	0 1 3 5 5
NK	0 2 0	2 1 5	X1X	0 2 0 2 2 2	0 1 5 3 5 5	Npsk0	1	2
NX	0 2 0	0 1 5	XX1	0 0 4 4 4	0 1 3 5 5	Kpsk0	1	2
XN	0 0 2 0	0 1 1 5	X1X1	0 0 0 4 4 4	0 1 3 3 5 5	Xpsk1	1	2
XK	0 2 4 4 4	2 1 5 5 5	K1N	0 0 2 0	0 1 1 5	NNpsk0	1 1 1 1	2 3 3 3
XX	0 2 4 4 4	0 1 5 5 5	K1K	0 4 4 4 4	2 1 5 5 5	NNpsk2	0 1 1 1	0 3 3 3
KN	0 0 2 0	0 3 1 5	KK1	0 4 4 4	0 3 5 5	NKpsk0	1 4 1 4	2 5 3 5
KK	1 4 4 4	2 4 5 5	K1K1	0 4 4 4	0 1 5 5 5	NKpsk2	0 4 1 4	0 3 3 5
KX	0 4 4 4	0 3 5 5	K1X	0 4 4 4 4	0 1 5 5 5	NXpsk2	0 4 1 4	0 3 3 5
IN	0 0 2 0	0 3 1 5	KX1	0 0 4 4 4	0 3 3 5 5	XNpsk3	0 0 4 1 4	0 1 3 3 5
IK	1 4 4 4	2 4 5 5	K1X1	0 0 4 4 4	0 1 3 5 5	XKpsk3	0 0 4 4 4	0 1 3 5 5
IX	0 4 4 4	0 3 5 5	I1N	0 0 2 0 2	0 1 1 5 1	KNpsk0	1 1 4 1	2 3 5 3
NK1	0 2 0	0 1 5	I1K	0 4 4 4 4	2 1 5 5 5	KNpsk2	0 1 4 1	0 3 5 3
NX1	0 0 0 2 0	0 1 3 1 5	IK1	0 4 4 4	0 3 5 5	INpsk1	1 1 4 1	2 3 5 3

- Analysis of 50+ Noise Handshake Patterns.
- We contribute a formally verified set of groundings for all security goals.
- We show that if pattern validity rules are not followed, subtle attacks can be found.

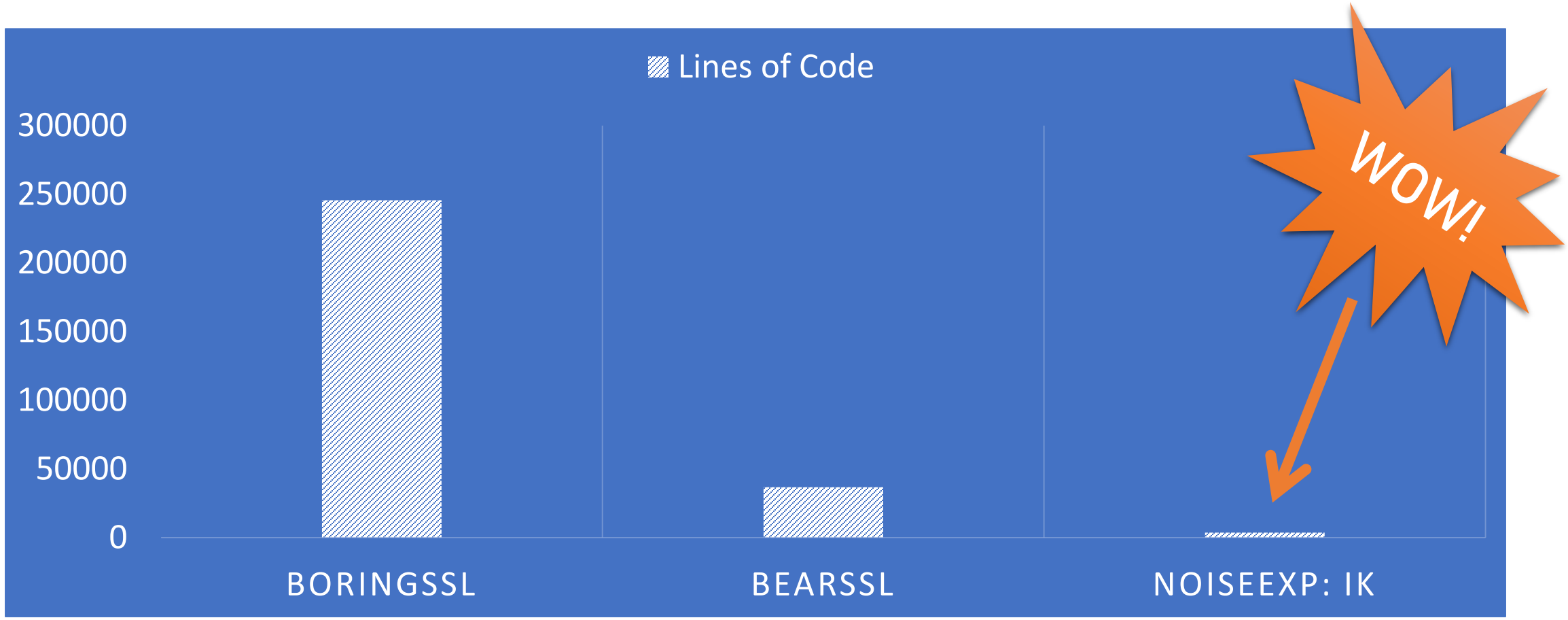
Contributions to Noise Specification

Pattern	Auth.	Conf.	Pattern	Auth.	Conf.	Pattern	Auth.	Conf.
N	0	2	X1N	0 0 0 0 2	0 1 1 3 1	I1K1	0 4 4 4 4	0 1 5 5 5
K	1	2	X1K	0 2 0 4 4 4	2 1 5 3 5 5	I1X	0 4 4 4 4	0 1 5 5 5
X	1	2	XK1	0 2 4 4 4	0 1 5 5 5	IX1	0 0 4 4 4	0 3 3 5 5
NN	0 0 0	0 1 1	X1K1	0 2 0 4 4 4	0 1 5 3 5 5	I1X1	0 0 4 4 4	0 1 3 5 5
NK	0 2 0	2 1 5	X1X	0 2 0 2 2 2	0 1 5 3 5 5	Npsk0	1	2
NX	0 2 0	0 1 5	XX1	0 0 4 4 4	0 1 3 5 5	Kpsk0	1	2
XN	0 0 2 0	0 1 1 5	X1X1	0 0 0 4 4 4	0 1 3 3 5 5	Xpsk1	1	2
XK	0 2 4 4 4	2 1 5 5 5	K1N	0 0 2 0	0 1 1 5	NNpsk0	1 1 1 1	2 3 3 3
XX	0 2 4 4 4	0 1 5 5 5	K1K	0 4 4 4 4	2 1 5 5 5	NNpsk2	0 1 1 1	0 3 3 3
KN	0 0 2 0	0 3 1 5	KK1	0 4 4 4	0 3 5 5	NKpsk0	1 4 1 4	2 5 3 5
KK	1 4 4 4	2 4 5 5	K1K1	0 4 4 4	0 1 5 5 5	NKpsk2	0 4 1 4	0 3 3 5
KX	0 4 4 4	0 3 5 5	K1X	0 4 4 4 4	0 1 5 5 5	NXpsk2	0 4 1 4	0 3 3 5
IN	0 0 2 0	0 3 1 5	KX1	0 0 4 4 4	0 3 3 5 5	XNpsk3	0 0 4 1 4	0 1 3 3 5
IK	1 4 4 4	2 4 5 5	K1X1	0 0 4 4 4	0 1 3 5 5	XKpsk3	0 0 4 4 4	0 1 3 5 5
IX	0 4 4 4	0 3 5 5	I1N	0 0 2 0 2	0 1 1 5 1	KNpsk0	1 1 4 1	2 3 5 3
NK1	0 2 0	0 1 5	I1K	0 4 4 4 4	2 1 5 5 5	KNpsk2	0 1 4 1	0 3 5 3
NX1	0 0 0 2 0	0 1 3 1 5	IK1	0 4 4 4	0 3 5 5	INpsk1	1 1 4 1	2 3 5 3

Improvements to Revision 34:

- More well-defined pattern validity rules and security grades.
- Higher assurance for fundamental pattern security grades.
- New security grades for all 23 deferred patterns.

Noise Versus TLS: Lines of Code



Time for a Demonstration!

Aspects that will be demonstrated:

1. Pattern designer and validator: https://noiseexplorer.com/
2. Automatically generated formal verification results: https://noiseexplorer.com/patterns/IK/ (as an example)
3. Detailed analysis results: https://noiseexplorer.com/patterns/IK/A.html (as an example)

The Future of Noise

Small, Use-Case Specific Protocols

- Entire library is ~1,000 LoC, specific Handshake Patterns can be smaller. (Great post by David Wong: https://cryptologie.net/article/446/quic-crypto-and-simple-state-machines/)
- Much smaller and more use-case specific state machine than TLS or similar.

Upcoming Work in Noise

- Signatures.
- Stateful hashing and symmetric crypto overhaul.
- NoiseSocket, NLS.
- Implementations that generate implementations?

Conclusion

Noise Explorer's potential: the ultimate online compendium for reasoning about, designing, studying, implementing and verifying Noise Handshake Patterns.

Special thanks: Bruno Blanchet, Trevor Perrin.

Related work: Benjamin Lipp, WireGuard verification in CryptoVerif.

Noise Explorer: https://noiseexplorer.com

Paper: https://eprint.iacr.org/2018/766