Cryptographic Protocol Verification

in AWS

Adam Petcher

apetcher@amazon.com

September 2020

Contents

- Cryptography Development in AWS
- Formal Verification of Cryptography
- Examples
    - Hybrid Key Encapsulation
    - SigV4

Cryptography Development

in AWS

Cryptography Development Purposes

- Service-Independent Protocols
    - e.g. Signature Version 4
- Implementation of Services
    - e.g. Key Management Service
- Custom Hardware
    - E.g. Nitro
- Standards
    - e.g. post-quantum, IoT
- Reusable Tools and Components
    - e.g. Encryption SDK

Ensuring the Security of AWS Cryptography

- Best practices and expert review
- Mathematical analysis and security proofs
- Formal verification

Formal Verification of Cryptography

How to Formally Verify Cryptography

- Machine-checked security proof
    - Provides additional assurance that proof is correct
- Ensures that system has some security property
- Carefully state capability of adversary
- Various models/approaches
    - Symbolic: primitives are perfect, ensure no “bad paths”
    - Computational: complexity-theoretic reduction

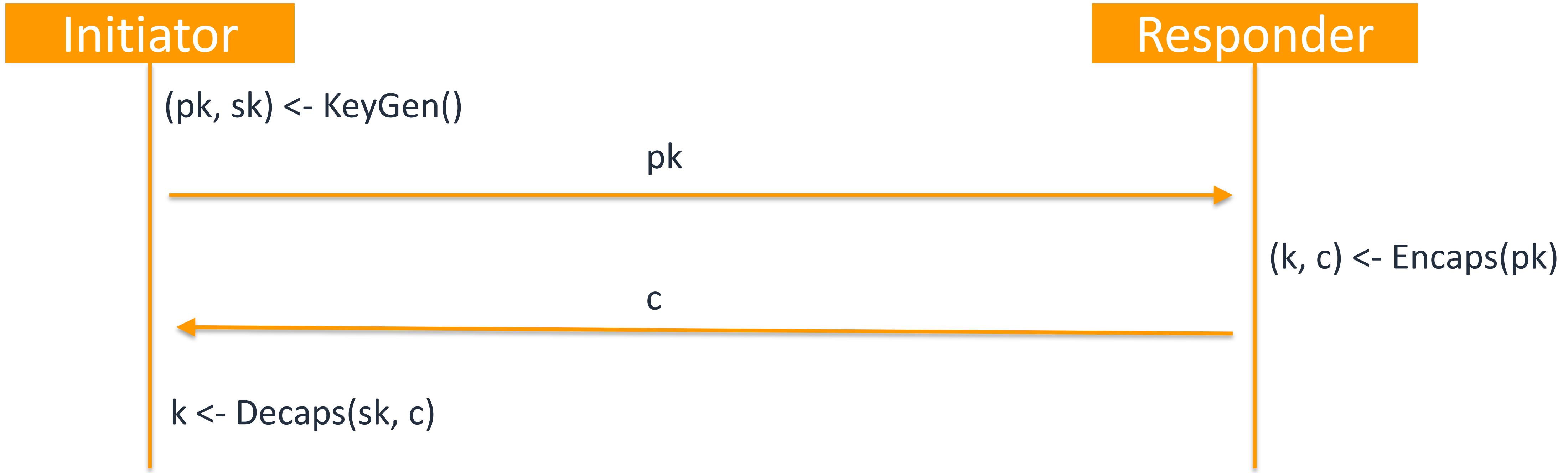
Why Not Stop at Paper Proofs?

- Sometimes paper proofs are good enough
    - Machine-checked proof can be expensive
- Significant proof flaws in the past
    - GCM: Error in lemma that bounds probability of collision
    - BCTV14 (Zcash): Error in lemma allowed counterfeiting
    - OCB2: Assumption applied incorrectly---completely insecure
- Machine-checked proofs can prevent expensive flaws

Example: Hybrid Key Encapsulation

Key Encapsulation Mechanisms

- Use public key cryptography to establish session keys
- E.g. Diffie-Hellman, RSA key transport



Hybrid Key Encapsulation

- Combine Multiple KEMs and achieve security of "strongest" one
    - Strength of KEM depends on adversary
    - Can combine classical and post-quantum KEM
- Concatenation KDF (CtKDF)
    - $(k_1, k_2, \ldots, k_n)$ produced by independent KEMs
    - $k \leftarrow \mathrm{HKDF}(k_1 \,||\, k_2 \,||\, \ldots \,||\, k_n$, label, context, length$)$
    - context includes all public information exchanged
    - Used in draft ETSI, NIST, and IETF standards.

Hybrid KEM Security

- IND-CPA security
  - Attacker sees public information in KEM exchanges
  - Attacker cannot distinguish resulting key from random
- CtKDF is IND-CPA secure assuming:
  - At least one underlying KEM is IND-CPA secure
  - HKDF is a secure KDF
- Proof is ”obvious”, but there are areas of concern
  - Is concatenation sufficient, or do we need to partition?
  - What information needs to go in context?
  - What distribution does HKDF need to extract? Is salt necessary?
  - Precise bound on adversary distinguishing key?

Formally Verified Hybrid KEM Security

- Machine-checked proof in computational model
- Complexity-theoretic reduction
    - Games define security definitions and assumptions
    - Proof is sequence of relations (e.g. equivalence) on pairs of games
    - Attacker can defeat KEM -> hardness assumption violated
- Proofs completed in Foundational Cryptography Framework (FCF)
    - Library for Coq proof assistant, inspired by EasyCrypt
    - Adds probability, relational reasoning, crypto definitions/arguments
    - Gives concrete numeric bounds on adversary success probability
    - No built-in complexity classes---allows quantum adversary/reduction

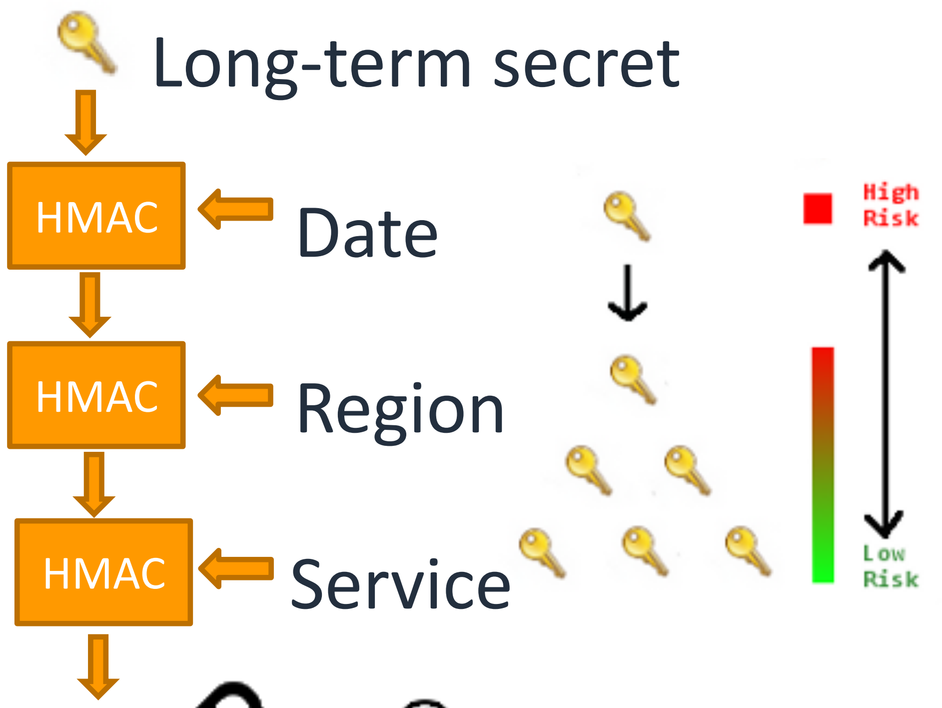
CtKDF Security Proofs

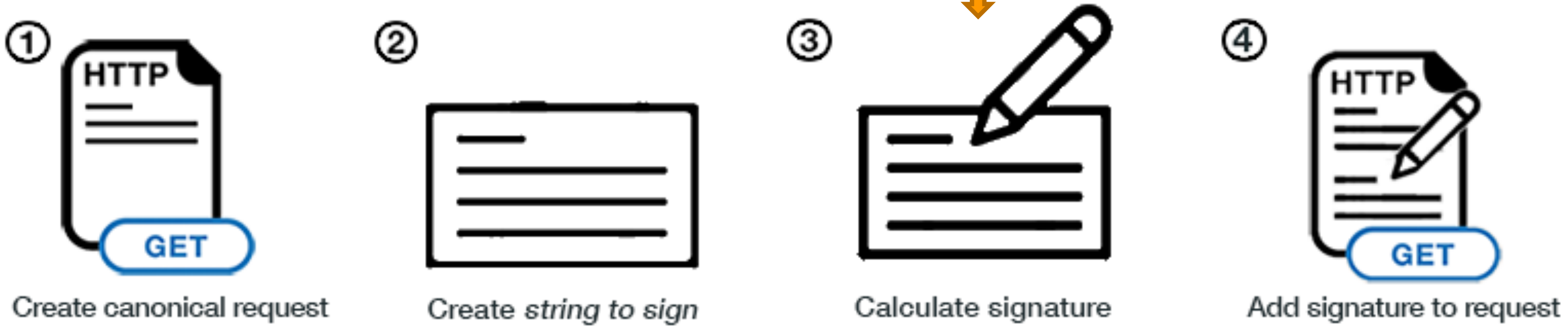
- IND-CPA in the standard model assuming:
  - At least one underlying KEM is IND-CPA secure
  - HKDF is secure KDF when extracting from a particular source:
    - X || Y || Z where Y is drawn from distribution of secure KEM, X and Z are anything
    - Source-specific assumption needed because KDF is not salted
- IND-CPA in the random oracle model assuming:
  - At least one underlying KEM is OW-CPA secure

Example: Signature Version 4

Signature Version 4 (SigV4)

- Used to authenticate all external AWS requests
- Signing key is derived from long-term secret, date, region, service
- Prevents exposure of long-term secret
- Reduces impact of exposure of short-term, local secrets





SigV4 Security Proof

- Goal: SigV4 is a secure MAC even when “unrelated” keys are compromised
- Stronger: SigV4 is a PRF even when “unrelated” keys are compromised
    - PRF: Pseudorandom Function---signatures appear random
- Universal Composability style: adversary cannot distinguish real/ideal
    - Real SigV4 functionality holds root secret
    - Ideal functionality returns random values for all new signatures
    - UC style is convenient for modeling compromise of secrets
- Adversary may (in any order, and any number of times)
    - Compromise a derived secret
    - Request a signature under an uncompromisable derived secret

UC-style Proof Mechanization in Quivela

- FCF is not well-suited for UC-style proofs
- Quivela: library for Coq proof assistant, in development
    - Earlier prototype: https://github.com/awslabs/quivela
- Checks UC-style security proofs
    - Functionalities defined in OO style with classes and objects
    - Objects can invoke methods on other objects
- Axiomatic semantics determines program behavior
    - Program logic for determining the behavior of single execution
    - Relational program logic for relating pairs of executions
- Semantics requires all programs are PPT, ignores negligible outcomes

Mechanized SigV4 Proof in Quivela

- Iterated PRF -> PRF on “disjoint” lists
    - “disjoint”: no list is strict prefix of the other
    - By induction on the max size of the list
- SigV4 Security
    - Main result: tags for uncompromisable keys are indistinguishable from random (chosen by RF)
    - Proof ensures that PRF is only called on “disjoint” lists

Summary

Summary

- AWS uses formal verification to increase assurance of security
- Cryptographic algorithms/protocols are verified via mechanized proof
- Using existing tools: EasyCrypt, FCF
- Developing new tools: Quivela
- See also: KMS proof in EasyCrypt (https://eprint.iacr.org/2019/1042.pdf)
- In case you have more questions: apetcher@amazon.com