slides from presentation at Real World Crypto 2019

Google

Tink: a cryptographic library

Bartosz Przydatek

joint work with **Daniel Bleichenbacher** and **Thai Duong**

with contributions by

Haris Andrianakis, **Thanh Bui**, **Thomas Holenstein**, **Charles Lee**, **Erhan Nergiz**, **Quan Nguyen**, **Veronika Slívová**, and others

Motivation

- cryptography is useful...
- ... but often difficult to use correctly
- complex APIs need in-depth expertise to be used safely
- focus of non-crypto developers is usually not on crypto
- simple mistakes can have serious consequences

Motivation: complex APIs: OpenSSL

```
int EVP_EncryptInit_ex(
    EVP_CIPHER_CTX *ctx,  const EVP_CIPHER *type,
    ENGINE *impl, unsigned char *key, unsigned char *iv);

int EVP_EncryptUpdate(
    EVP_CIPHER_CTX *ctx,  unsigned char *out,
    int *outl, const unsigned char *in, int inl);

int EVP_EncryptFinal_ex(
    EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl);
```

Motivation: complex APIs: OpenSSL

```
int EVP_EncryptInit_ex(
    EVP_CIPHER_CTX *ctx,  const EVP_CIPHER *type,
    ENGINE *impl, unsigned char *key, unsigned char *iv);

int EVP_EncryptUpdate(
    EVP_CIPHER_CTX *ctx,  unsigned char *out,
    int *outl, const unsigned char *in, int inl);

int EVP_EncryptFinal_ex(
    EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl);
```

Motivation: complex APIs: Crypto API NG

```
NTSTATUS BCryptEncrypt(
    BCRYPT_KEY_HANDLE hKey,
    PUCHAR            pbInput,
    ULONG             cbInput,
    VOID              *pPaddingInfo,
    PUCHAR            pbIV,
    ULONG             cbIV,
    PUCHAR            pbOutput,
    ULONG             cbOutput,
    ULONG             *pcbResult,
    ULONG             dwFlags
);
```

Motivation: complex APIs: Java JCE

```
SecureRandom secureRandom = new SecureRandom();
byte[] key = new byte[16];
secureRandom.nextBytes(key);
SecretKey secretKey = SecretKeySpec(key, "AES");

byte[] iv = new byte[IV_SIZE];
secureRandom.nextBytes(iv);
GCMParameterSpec parameterSpec = new GCMParameterSpec(128, iv);

Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
cipher.init(Cipher.ENCRYPT_MODE, secretKey, parameterSpec);

// continued...
```

Motivation: complex APIs: Java JCE

```
SecureRandom secureRandom = new SecureRandom();
byte[] key = new byte[16];
secureRandom.nextBytes(key);
SecretKey secretKey = SecretKeySpec(key, "AES");

byte[] iv = new byte[IV_SIZE];
secureRandom.nextBytes(iv);
GCMParameterSpec parameterSpec = new GCMParameterSpec(128, iv);

Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
cipher.init(Cipher.ENCRYPT_MODE, secretKey, parameterSpec);

// continued...
```

Motivation: complex APIs: Java JCE (cont.)

```
// continued...

byte[] ciphertext = new byte[IV_SIZE + plaintext.length + TAG_SIZE];
System.arraycopy(iv, 0, ciphertext, 0, IV_SIZE);
if (associatedData != null) {
  cipher.updateAAD(associatedData);
}
cipher.doFinal(plaintext, 0, plaintext.length, ciphertext, IV_SIZE);
return ciphertext;
```

Motivation: complex APIs: Java JCE (cont.)

```
// continued...

byte[] ciphertext = new byte[IV_SIZE + plaintext.length + TAG_SIZE];
System.arraycopy(iv, 0, ciphertext, 0, IV_SIZE);
if (associatedData != null) {
  cipher.updateAAD(associatedData);
}
cipher.doFinal(plaintext, 0, plaintext.length, ciphertext, IV_SIZE);
return ciphertext;
```

Motivation: ambiguous yet inextensible APIs

C++ Keyczar: Keyczar object can do “everything”

```
class Keyczar {
  virtual bool Sign(...);
  virtual bool AttachedSign(...);
  virtual bool Verify(...);
  virtual bool AttachedVerify(...);
  virtual bool Encrypt(...);
  virtual bool Decrypt(...);
  // ...
  virtual bool IsAcceptablePurpose(KeyPurpose purpose);
}
```

… yet this might still be not enough!

Motivation: ambiguous yet inextensible APIs

Java Keyczar: one Encrypter for all encryption

```
public class Encrypter extends Keyczar {

  public byte[] encrypt(byte[] input)  {  /*...*/ }

  @Override boolean isAcceptablePurpose(KeyPurpose purpose)
}
```

- Mixes public-key encryption and numerous flavours of symmetric encryption
- Bound to a global KeyPurpose-enum

Outline

- **Tink design goals**
- **User's perspective**: primitives and keyset handles
- **Tink core**: keys, key managers, keysets, registry
- **Key management** features
- **Readability & Auditability**: security guarantees and configs
- **Extensibility**: custom implementations & custom primitives
- Current **status** and future **plans**

Tink design goals

- **Security**
  - hard-to-misuse API
  - reuse of proven and well-tested libraries (project Wycheproof)
- **Usability**
  - simple & easy-to-use API
  - user can focus on the desired functionality

Tink design goals (cont.)

- **Readability and Auditability**
  - functionality “visible” in code,
  - control over employed cryptographic schemes
- **Extensibility**
  - easy to add new functionalities, schemes, formats
  - support for local customizations

Tink design goals (cont.)

- **Agility**
  - built-in key rotation
  - support for deprecation of obsolete/broken schemes
- **Interoperability**
  - available in many languages and on many platforms
  - integration with external services (e.g. KMS)

User's perspective: Primitives

Primitive: an abstract representation of a crypto functionality

- defines functionality in a form of an interface
- not bound to any specific implementation or a global enum
- (official) implementations come with security guarantees

User's perspective: MAC primitive

Message Authentication Code (MAC)

```
public interface Mac {
  byte[] computeMac(final byte[] data) throws ...
  void verifyMac(final byte[] mac, final byte[] data) throws...
}
```

User's perspective: AEAD primitive

Authenticated Encryption with Associated Data (AEAD)

```
public interface Aead {
  byte[] encrypt(final byte[] plaintext, final byte[] associatedData)
      throws...
  byte[] decrypt(final byte[] ciphertext, final byte[] associatedData)
      throws...
}
```

User's perspective: Streaming AEAD primitive

```
public interface StreamingAead {
 OutputStream newEncryptingStream(OutputStream ciphertextDestination,
                                  byte[] associatedData) throws...
 InputStream newDecryptingStream(InputStream ciphertextSource,
                                 byte[] associatedData) throws...
 /* ... */
}
```

User's perspective: AEAD primitive in action

```
import com.google.crypto.tink.Aead;
import com.google.crypto.tink.KeysetHandle;


// 1. Generate or retrieve the key material.
KeysetHandle keysetHandle = ...;

// 2. Get the primitive.
Aead aead = keysetHandle.getPrimitive(Aead.class);

// 3. Use the primitive to encrypt a plaintext,
byte[] ciphertext = aead.encrypt(plaintext, aad);
```

User's perspective: AEAD primitive in action

```
import com.google.crypto.tink.Aead;
import com.google.crypto.tink.KeysetHandle;
import com.google.crypto.tink.aead.AeadKeyTemplates;

// 1. Generate ~~or retrieve~~ the key material.
KeysetHandle keysetHandle =
                    KeysetHandle.generateNew(AeadKeyTemplates.AES128_GCM);
// 2. Get the primitive.
Aead aead = keysetHandle.getPrimitive(Aead.class);

// 3. Use the primitive to encrypt a plaintext,
byte[] ciphertext = aead.encrypt(plaintext, aad);
```

User's perspective: AEAD primitive in action

```
import com.google.crypto.tink.Aead;
import com.google.crypto.tink.KeysetHandle;
import com.google.crypto.tink.integration.android.AndroidKeysetManager;

// 1. Generate or retrieve the key material.
AndroidKeysetManager keysetManager = AndroidKeysetManager.Builder()...;
KeysetHandle keysetHandle = keysetManager.getKeysetHandle();
// 2. Get the primitive.
Aead aead = keysetHandle.getPrimitive(Aead.class);

// 3. Use the primitive to encrypt a plaintext,
byte[] ciphertext = aead.encrypt(plaintext, aad);
```

Tink core: keys

Key: a container for cryptographic key material and params

- identified by a string: **key type** (a.k.a. **type url**), e.g.

  `"type.googleapis.com/google.crypto.tink.AesGcmKey"`

- implemented as a protocol buffer:

```
message AesGcmKey {
    uint32 version;
    bytes key_value;
}
```

Tink core: key managers

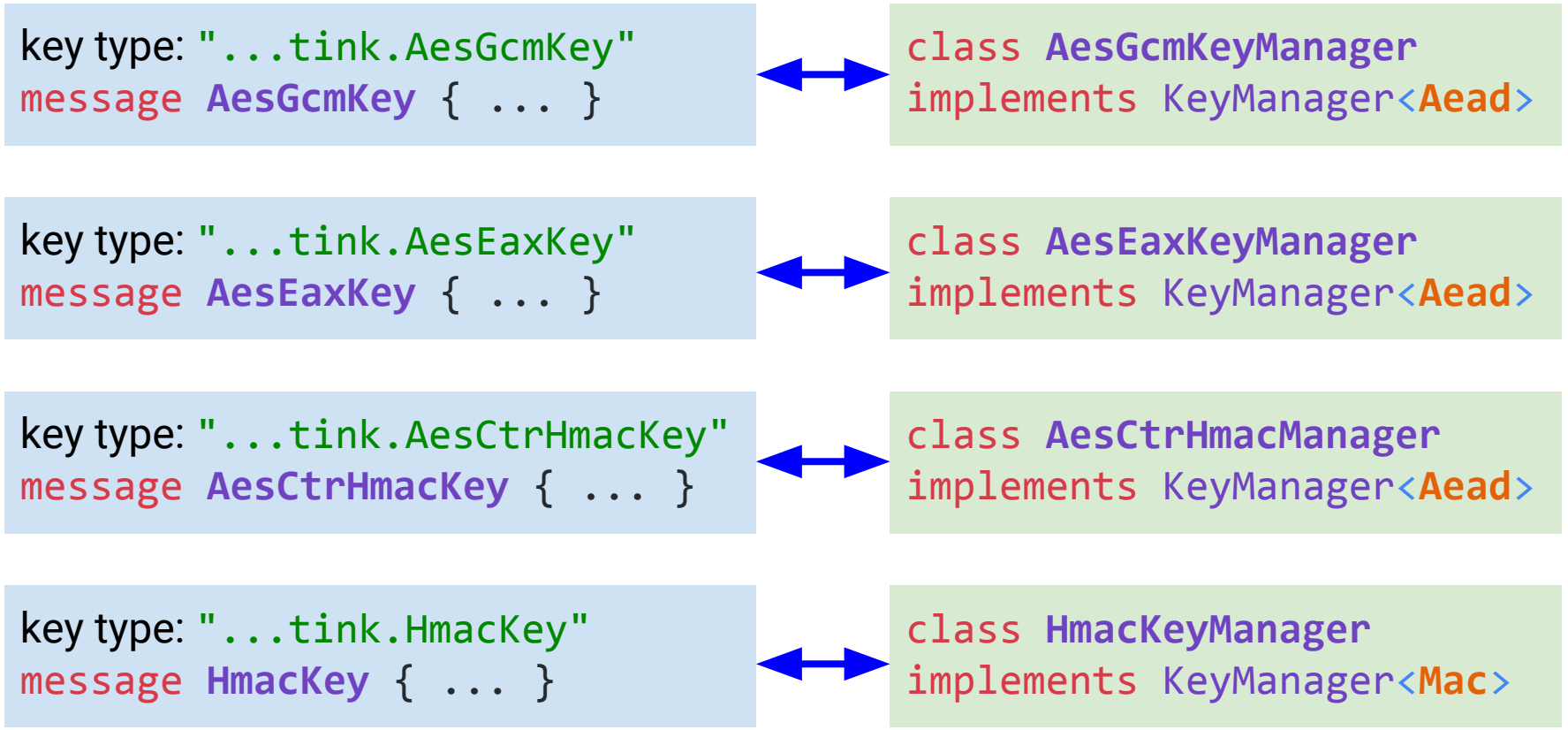
Key Manager: a manager for keys of a specific **key type**, "knows" which **primitive** corresponds to the key type, e.g.

```
class AesGcmKeyManager implements KeyManager<Aead> {
  @Override
  public Aead getPrimitive(aesGcmKey) {...};

  @Override
  public AesGcmKey newKey(aesGcmKeyFormat) {...};

  /* ... */
}
```

Tink core: keys and key managers



Tink core: keyset and keyset handle

- **Keyset**: a collection of keys
  - all keys in a keyset correspond to a single primitive
  - primary tool for key rotation
- **Keyset Handle**: a wrapper around a Keyset
  - restricts access to key material and other sensitive data

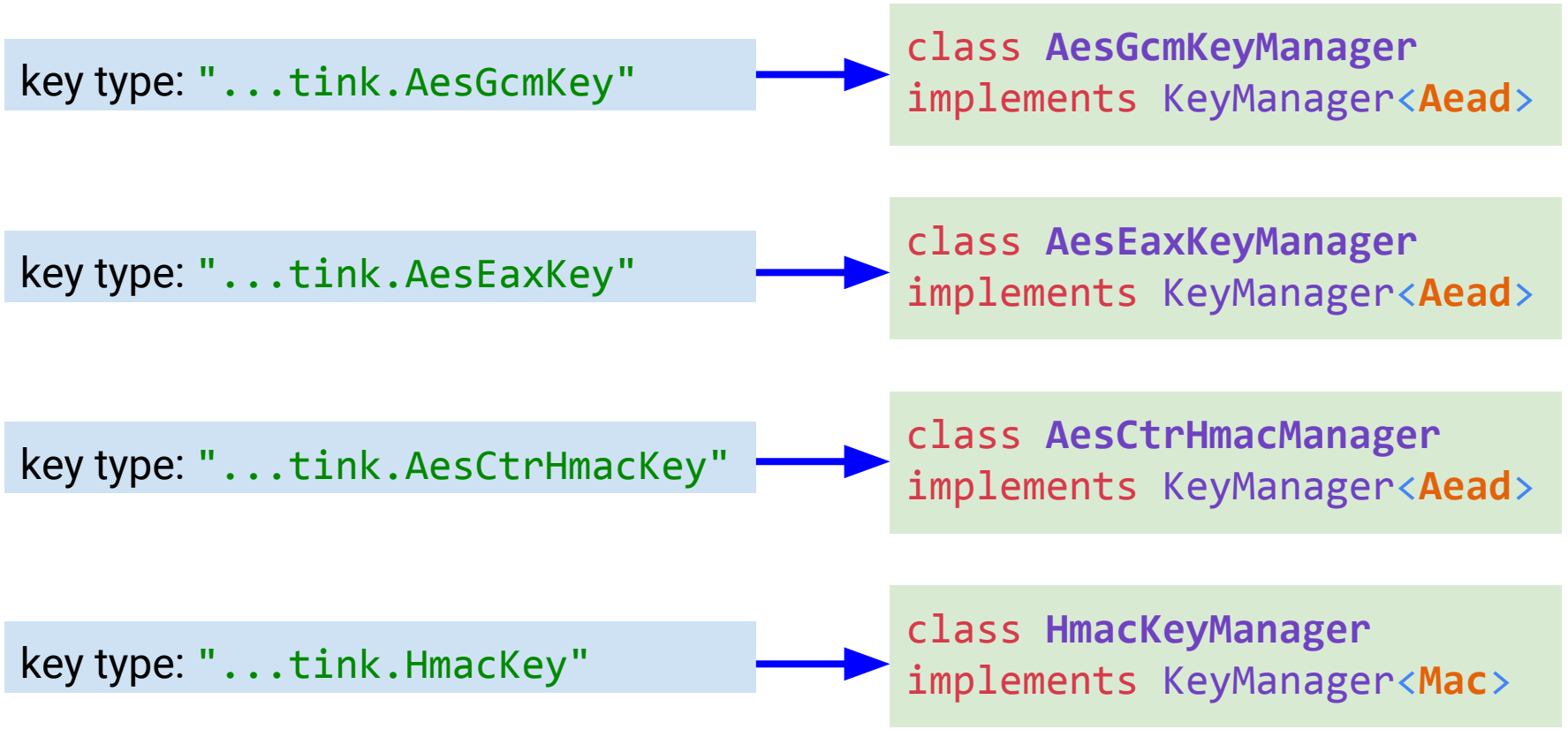
Tink core: keyset and keyset handle example

Tink core: Registry

Registry: a container for key managers used by an application

- A mapping from key type to a key manager object
- Initialized at startup
  - automatically: `TinkConfig.register()`
  - .. or manually: `Registry.registerKeyManager(...)`
- The foundation of obtaining Primitives
  - indirectly via `KeysetHandle.getPrimitive(...)`
  - or directly: `Registry.getPrimitive(...)`

Tink core: Registry



Key management features: key rotation

Key rotation via keysets

- a distinguished primary key for creation of crypto data (ciphertexts, signatures, …)
- matching of crypto data with a suitable key in a keyset
- disabling of obsolete keys

Key management features (cont.)

- Uniform handling of external keys (KMS, HSM, …)
  - “key” in a keyset contains only a reference to KMS
  - a keyset can contain both external and regular keys
- Gradual deprecation of cryptographic schemes
  - can forbid creation of new keys of deprecated schemes

Readability & Auditability

- Implementations of Primitives guarantee properties

```
Aead aead = handle1.getPrimitive(Aead.class);
byte[] ciphertext1 = aead.encrypt(plaintext1, associatedData);

HybridEncrypt hybridEncrypt = handle2.getPrimitive(HybridEncrypt.class);
byte[] ciphertext2 = hybridEncrypt.encrypt(plaintext2, contextInfo);
```

- Registry and Configs
  - full control over Primitives and their implementations
  - stats about usage of cryptographic schemes (planned)

Extensibility

- Custom key types and implementations of Tink primitives
- Definition and implementation of custom primitives
- Registry, keysets, key rotation, etc. work as with standard components

Extensibility: custom implementation of AEAD

- Define custom key type

  `type.googleapis.com/my.org.MyCustomKey`

```
message MyCustomKey {                 |    message MyCustomKeyFormat {
  uint32 version;                     |      // params for generating new keys
  // custom fields and params         |    }
}                                     |
```

- Implement key manager for the custom key type

```
class MyCustomKeyManager
extends KeyManagerBase<Aead, MyCustomKey, MyCustomKeyFormat> {...}
```

- Register the custom key manager.

Extensibility: custom primitives

- Define the interface of the custom primitive

```
public interface MyPrimitive {
  byte[] computeSomeCryptoData(final byte[] input)
      throws GeneralSecurityException;
}
```

- Implement a primitive wrapper and register it

```
class MyPrimitiveWrapper implements PrimitiveWrapper<MyPrimitive> {
  @Override
  public MyPrimitive wrap(final PrimitiveSet<MyPrimitive> pset);
}
```

- Implement key manager(s) & use them as for Tink primitives

Current status and future plans

Tink is open-sourced on GitHub: github.com/google/tink

- Supported Primitives:
  - Message Authentication Codes (MAC)
  - Authenticated Encryption with Associated Data (AEAD)
  - Deterministic AEAD
  - Streaming AEAD
  - Digital Signatures: PublicKeySign and PublicKeyVerify
  - Hybrid Encryption: HybridEncrypt and HybridDecrypt

Current status and future plans (cont.)

- Supported languages
  - current: Java, C++, Objective C
  - in preparation: Go, JavaScript, Python
  - open-source community driven: PHP
- Integration with KMS offerings
  - **Java**: AWS KMS, Google Cloud KMS, Android Keystore
  - **Objective C**: Apple Keychain
  - **C++** (in preparation): AWS KMS, Google Cloud KMS

Summary

- Tink: crypto as a tool for non-crypto developers
- Multiple languages, multiple platforms
- Secure, simple, w/ key rotation, readable, extensible, …
- … and much more (not in the talk): thread safety, protections against side-channel attacks, efficiency, versioning, ...
- Open-source, external contributions are very welcome!