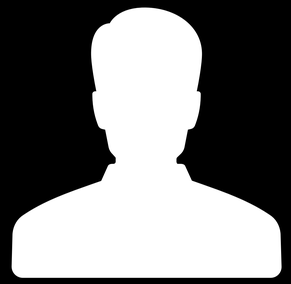
#WWDC19

Cryptography and Your Apps

Yannick Sierra, Security Engineering and Architecture
Frederic Jacobs, Security Engineering and Architecture

Protecting Users and Businesses

Protecting Users and Businesses

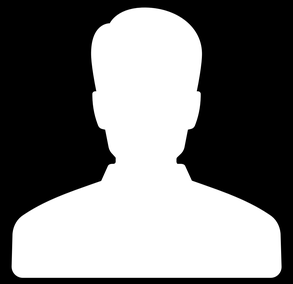


User

Profile

Data

Protecting Users and Businesses



User

Profile

Data

Business

Content

Account

Foundation of Apple Security and Privacy Features

High Stakes

Performance, Energy, Security

Cryptography Is a Tool

Cryptography Is a Tool

Authentication

Cryptography Is a Tool

Authentication

Encryption

Cryptography Is a Tool

Authentication

Encryption

Integrity

Cryptography Is a Tool

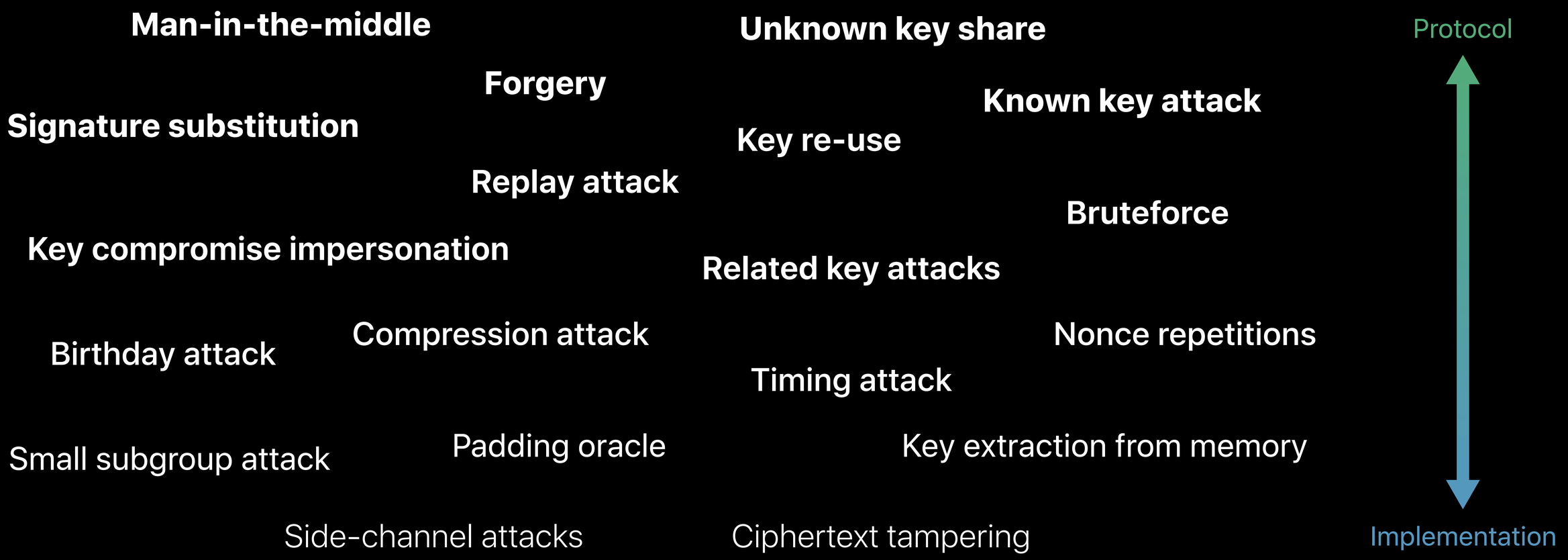
Authentication

Encryption

Integrity

Compose primitives into protocols

Cryptography Is Hard to Get Right



High Risks, High Effort

Design robust protocol

Monitor for new attacks

Fix Apps and dependencies

Benefit from native features

Common Cases Covered by the System

Common Cases Covered by the System

Protect data on device

Common Cases Covered by the System

Protect data on device

Protect credentials and keys

Common Cases Covered by the System

Protect data on device

Protect credentials and keys

Share data across devices and users

Common Cases Covered by the System

Protect data on device

Protect credentials and keys

Share data across devices and users

Secure network connections

Common Cases Covered by the System

Protect data on device

Protect credentials and keys

Share data across devices and users

Secure network connections

Verify remote parties

Protect Data on Device

Protect Data on Device

- Advice from various websites

Protect Data on Device

- ✗ Advice from various websites
- ✓ Use data protection
  - Passcode protection and Secure Enclave
  - Thwart brute-force
  - Post compromise recovery
  - File protection types

apple.com/business/site/docs/iOS_Security_Guide.pdf

Protect Data on Device

- ✗ Advice from various websites
- ✓ Use data protection
  - Passcode protection and Secure Enclave
  - Thwart brute-force
  - Post compromise recovery
  - File protection types

  From **Until First Authentication**
  to **Complete**

apple.com/business/site/docs/iOS_Security_Guide.pdf

```swift
// Write file, available when device unlocked
do {
  try data.write(to: fileURL, options: .completeFileProtection)
}
catch {
  // Handle errors
}
```

```swift
// Write file, available when device unlocked
do {
  try data.write(to: fileURL, options: .completeFileProtection)
}
catch {
  // Handle errors
}
```

Protect Credentials and Keys

Protect Credentials and Keys

- ✕ Write credentials in defaults, or files

Protect Credentials and Keys

- Write credentials in defaults, or files
- Use Keychain (SecItem)
  - Local or iCloud Keychain
  - All the file protection classes + extras

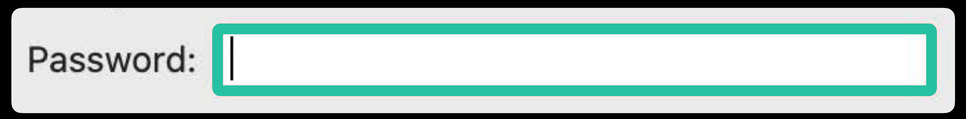
Protect Credentials and Keys

- Write credentials in defaults, or files
- Use Keychain (SecItem)
  - Local or iCloud Keychain
  - All the file protection classes + extras
- Use LocalAuthentication
  - Protect operations with policies, for example protect access to a key with Face ID

New LocalAuthentication Policies

User authentication on macOS

```
LAPolicyDeviceOwnerAuthentication
```

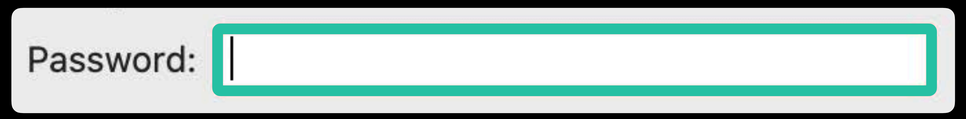


New LocalAuthentication Policies

NEW

User authentication on macOS

```
LAPolicyDeviceOwnerAuthentication
```



New LocalAuthentication Policies

NEW

User authentication on macOS

```
LAPolicyDeviceOwnerAuthentication
```

Password:

Apple Watch or Biometrics NEW

```
LAPolicyDeviceOwnerAuthenticationWithBiometricsOrWatch
```

New LocalAuthentication Policies

NEW

User authentication on macOS

```
LAPolicyDeviceOwnerAuthentication
```

Apple Watch or Biometrics NEW

```
LAPolicyDeviceOwnerAuthenticationWithBiometricsOrWatch
```

Apple Watch only NEW

```
LAPolicyDeviceOwnerAuthenticationWithWatch
```

Share Data Across Devices and Users

Share Data Across Devices and Users

- Encrypt assets in Private CloudKit Database
  - No need for application user sign in
  - Apple as a trusted party
  - iCloud identities and access control

```
// Save the contents of a given fileURL to a per-user location
let asset = CKAsset(fileURL: ...)
let record: CKRecord = ...
record["AssetField"] = asset
let database = CKContainer.default().privateCloudDatabase
database.save(record) { ... }
```

Secure Network Connection

Secure Network Connection

- Use custom protocol to server
  Secure Transport

Secure Network Connection

- Use custom protocol to server
  Secure Transport

TLS 1.3

- Network framework with default TLS
  URLSession with App Transport Security
  - Defaults guarantee strong security and great performance

developer.apple.com/videos/play/wwdc2018/715/

```swift
// With Network Framework
let conn = NWConnection(host: "imap.mail.me.com", port: .imaps, using: .tls)
conn.start(queue: .main)
```

```swift
// With URLSession
let url = URL(string: "https://www.apple.com")!
let task = URLSession.shared.dataTask(with: url) { (data, response, error) in
  if let error = error {
    // Handle error
  }
  // Operation on data
}
task.resume()
```

```swift
// With Network Framework
let conn = NWConnection(host: "imap.mail.me.com", port: .imaps, using: .tls)
conn.start(queue: .main)
```

```swift
// With URLSession
let url = URL(string: "https://www.apple.com")!
let task = URLSession.shared.dataTask(with: url) { (data, response, error) in
  if let error = error {
    // Handle error
  }
  // Operation on data
}
task.resume()
```

Verify Remote Parties

Verify Remote Parties

- Use for TLS
  Custom X.509/Certificate parser

Verify Remote Parties

- Use for TLS
  Custom X.509/Certificate parser
- SecTrust
  - Trust criteria defined by a policy
  - Validate certificate against policy
  - Use trusted keys to encrypt or verify data

```
// Evaluate a certificate validity asynchronously and with legible errors
SecTrustEvaluateAsyncWithError(trust, queue) { (trust, success, error) in
  if (success) {
    let publicKey = SecTrustCopyPublicKey(trust);
    // Use key…
  } else {
    // Handle errors
  }
}
```

NEW

```
// Evaluate a certificate validity asynchronously and with legible errors
SecTrustEvaluateAsyncWithError(trust, queue) { (trust, success, error) in
  if (success) {
    let publicKey = SecTrustCopyPublicKey(trust);
    // Use key…
  } else {
    // Handle errors
  }
}
```

NEW

System Features

System Features

Protect data on device

Protect keys and credentials

Share data across devices and users

Secure network connections

Verify remote parties

Use system frameworks

And

Other Needs

Other Needs

Interoperate between platforms

Other Needs

Interoperate between platforms

Authenticate with your service

Other Needs

Interoperate between platforms

Authenticate with your service

Implement a specification

Other Needs

Interoperate between platforms

Authenticate with your service

Implement a specification

Use Apple CryptoKit

NEW

Introducing Apple CryptoKit

Frederic Jacobs, Security Engineering and Architecture

Apple CryptoKit

Apple CryptoKit

Swift Framework

Apple CryptoKit

Swift Framework

Secure Algorithms

Apple CryptoKit

Swift Framework

Secure Algorithms

Secure Enclave

Apple CryptoKit

Swift Framework

Secure Algorithms

Secure Enclave

Performance

```
// Encrypting with a C crypto API
let status = ciphertext.withUnsafeMutableBytes { (cipherPtr: UnsafeMutableRawBufferPointer) in
    tag.withUnsafeMutableBytes { (tagPtr: UnsafeMutableRawBufferPointer) in
        nonceData.withUnsafeBytes { (noncePtr: UnsafeRawBufferPointer)  in
            key.withUnsafeBytes { (keyPtr: UnsafeRawBufferPointer) in
                plaintext.withUnsafeBytes { (plaintextPtr: UnsafeRawBufferPointer) in
                    return encrypt(keyPtr, plaintextPtr, noncePtr, keyPtr, cipherPtr, tagPtr)
                }
            }
        }
    }
}
```

```
// Encrypting with a C crypto API
let status = ciphertext.withUnsafeMutableBytes { (cipherPtr: UnsafeMutableRawBufferPointer) in
    tag.withUnsafeMutableBytes { (tagPtr: UnsafeMutableRawBufferPointer) in
        nonceData.withUnsafeBytes { (noncePtr: UnsafeRawBufferPointer)  in
            key.withUnsafeBytes { (keyPtr: UnsafeRawBufferPointer) in
                plaintext.withUnsafeBytes { (plaintextPtr: UnsafeRawBufferPointer) in
                    return encrypt(keyPtr, plaintextPtr, noncePtr, keyPtr, cipherPtr, tagPtr)
                }
            }
        }
    }
}



// Encrypting with Apple CryptoKit
let sealed = try AES.GCM.seal(dataToEncrypt, using: symmetricKey)
```

```
// Generating and releasing a cryptographic key for a C Crypto API
let keyByteCount = 256/8
```

```
// Generating and releasing a cryptographic key for a C Crypto API
let keyByteCount = 256/8
var key = Array(repeating: 0, count: keyByteCount)
let err = SecRandomCopyBytes(kSecRandomDefault, keyByteCount, &key)
```

```
// Generating and releasing a cryptographic key for a C Crypto API
let keyByteCount = 256/8
var key = Array(repeating: 0, count: keyByteCount)
let err = SecRandomCopyBytes(kSecRandomDefault, keyByteCount, &key)
if (err != errSecSuccess) {
  // Safely handle the error
}
```

```
// Generating and releasing a cryptographic key for a C Crypto API
let keyByteCount = 256/8
var key = Array(repeating: 0, count: keyByteCount)
let err = SecRandomCopyBytes(kSecRandomDefault, keyByteCount, &key)
if (err != errSecSuccess) {
  // Safely handle the error
}
// Use the Key
...
```

```
// Generating and releasing a cryptographic key for a C Crypto API
let keyByteCount = 256/8
var key = Array(repeating: 0, count: keyByteCount)
let err = SecRandomCopyBytes(kSecRandomDefault, keyByteCount, &key)
if (err != errSecSuccess) {
  // Safely handle the error
}
// Use the Key
...
// Zeroize the key
memset_s(&key, keyByteCount, 0, keyByteCount)
```

```
// Generating and releasing a cryptographic key for a C Crypto API
let keyByteCount = 256/8
var key = Array(repeating: 0, count: keyByteCount)
let err = SecRandomCopyBytes(kSecRandomDefault, keyByteCount, &key)
if (err != errSecSuccess) {
  // Safely handle the error
}
// Use the Key
...
// Zeroize the key
memset_s(&key, keyByteCount, 0, keyByteCount)



// Generating and releasing a cryptographic key with Apple CryptoKit
let key = SymmetricKey(size: .bits256)
```

Apple CryptoKit and Swift

Apple CryptoKit and Swift

Strongly typed interfaces

Apple CryptoKit and Swift

Strongly typed interfaces

Memory management

Apple CryptoKit and Swift

Strongly typed interfaces

Memory management

Equatable conformances

Apple CryptoKit and Swift

Strongly typed interfaces

Memory management

Equatable conformances

Generics

Apple CryptoKit

Swift Framework

Apple CryptoKit

Swift Framework

Secure Algorithms

Architecture

NEW

CryptoKit

Architecture

NEW

CryptoKit

Hash Functions

SHA-256
SHA-384
SHA-512

Architecture

NEW

CryptoKit

Hash Functions

SHA-256
SHA-384
SHA-512

Symmetric-Key Cryptography

Message Authentication Codes

HMAC

Architecture

NEW

CryptoKit

Hash Functions

SHA-256
SHA-384
SHA-512

Symmetric-Key Cryptography

Message Authentication Codes

HMAC

Authenticated Encryption

AES-GCM
Chacha20Poly1305

Architecture

NEW

CryptoKit

Hash Functions

SHA-256
SHA-384
SHA-512

Symmetric-Key Cryptography

Message Authentication Codes

HMAC

Authenticated Encryption

AES-GCM
Chacha20Poly1305

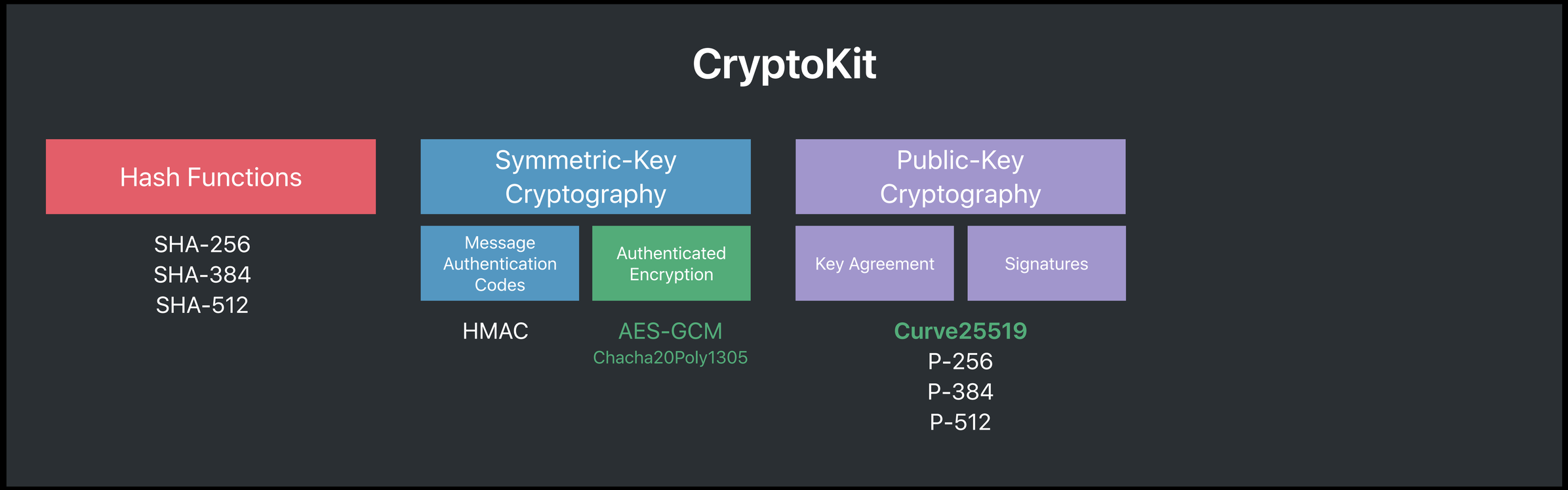
Public-Key Cryptography

Key Agreement

Curve25519
P-256
P-384
P-512

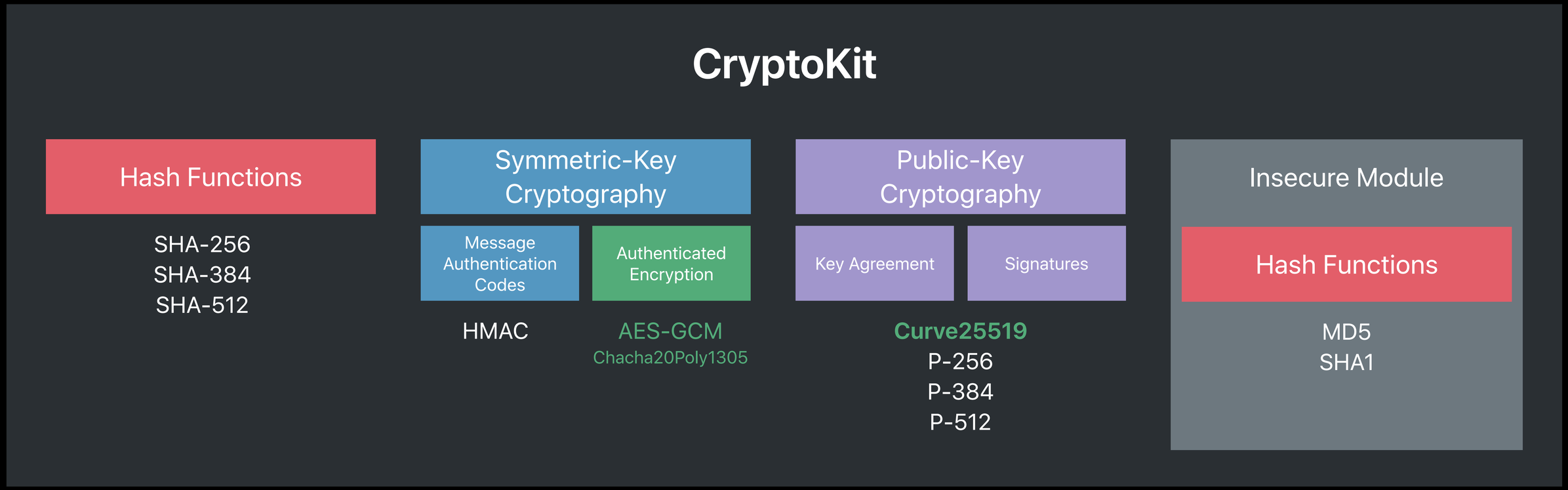
Architecture

NEW



Architecture

NEW



Hash Functions

Hash Functions

Produces deterministic fixed-size digest

Hash Functions

Produces deterministic fixed-size digest

Collision resistance

Verifying the Integrity of a File

```
let audioData = FileManager.default.contents(atPath: filePath)!
let digest = SHA256.hash(data: audioData)
```

Verifying the Integrity of a File

```
let audioData = FileManager.default.contents(atPath: filePath)!
let digest = SHA256.hash(data: audioData)
```

Hashing Data Incrementally

```
var hasher = SHA256()
let fileStream = InputStream(fileAtPath: filePath)!
fileStream.open()
let bufferSize = 64000
let buffer = UnsafeMutablePointer<UInt8>.allocate(capacity: bufferSize)

while fileStream.hasBytesAvailable {
    let read = fileStream.read(buffer, maxLength: bufferSize)
    let bufferPointer = UnsafeRawBufferPointer(start: buffer, count: read)
    hasher.update(bufferPointer: bufferPointer)
}

let digest = hasher.finalize()
```

Hashing Data Incrementally

```
var hasher = SHA256()
let fileStream = InputStream(fileAtPath: filePath)!
fileStream.open()
let bufferSize = 64000
let buffer = UnsafeMutablePointer<UInt8>.allocate(capacity: bufferSize)

while fileStream.hasBytesAvailable {
    let read = fileStream.read(buffer, maxLength: bufferSize)
    let bufferPointer = UnsafeRawBufferPointer(start: buffer, count: read)
    hasher.update(bufferPointer: bufferPointer)
}

let digest = hasher.finalize()
```

Hashing Data Incrementally

```
var hasher = SHA256()
let fileStream = InputStream(fileAtPath: filePath)!
fileStream.open()
let bufferSize = 64000
let buffer = UnsafeMutablePointer<UInt8>.allocate(capacity: bufferSize)

while fileStream.hasBytesAvailable {
    let read = fileStream.read(buffer, maxLength: bufferSize)
    let bufferPointer = UnsafeRawBufferPointer(start: buffer, count: read)
    hasher.update(bufferPointer: bufferPointer)
}

let digest = hasher.finalize()
```

Hashing Data Incrementally

```
var hasher = SHA256()
let fileStream = InputStream(fileAtPath: filePath)!
fileStream.open()
let bufferSize = 64000
let buffer = UnsafeMutablePointer<UInt8>.allocate(capacity: bufferSize)

while fileStream.hasBytesAvailable {
    let read = fileStream.read(buffer, maxLength: bufferSize)
    let bufferPointer = UnsafeRawBufferPointer(start: buffer, count: read)
    hasher.update(bufferPointer: bufferPointer)
}

let digest = hasher.finalize()
```

Authenticated Encryption

NEW

Authenticated Encryption

NEW

Provides both authentication and encryption

Authenticated Encryption

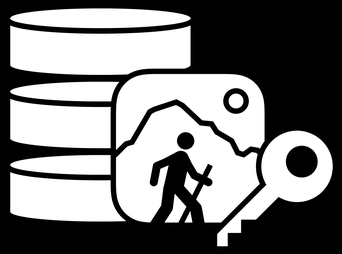
NEW

Provides both authentication and encryption

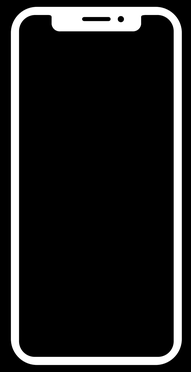
Authentication prevents a wide range of attacks

Example

Hiking app



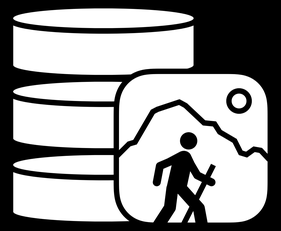
Hiking App
Server



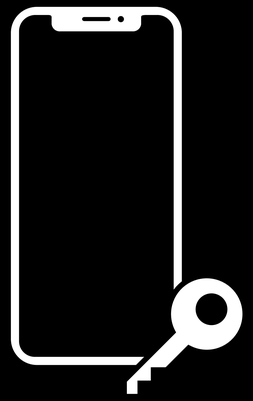
Content Delivery Network

Example

Hiking app



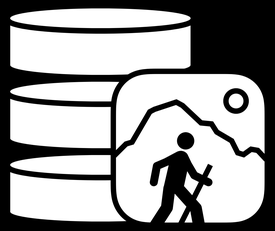
Hiking App
Server



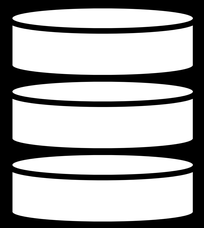
Content Delivery Network

Example

Hiking app



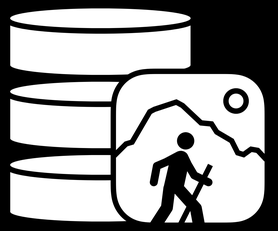
Hiking App
Server



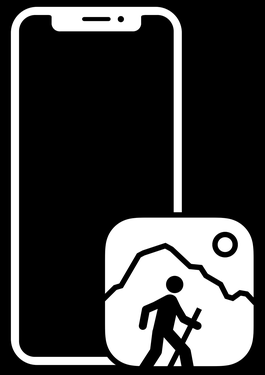
Content Delivery Network

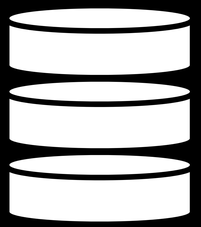
Example

Hiking app



Hiking App Server





Content Delivery Network

Decrypting Content

Decrypting Content

```
// Initialize the decryption key
let key = SymmetricKey(data: keyData)
```

Decrypting Content

```
// Initialize the decryption key
let key = SymmetricKey(data: keyData)

// Initialize the sealed box
guard let sealedBox = ChaChaPoly.SealedBox(combined: downloadedData) else {
    throw MapDownloaderError.invalidDownload
}
```

Decrypting Content

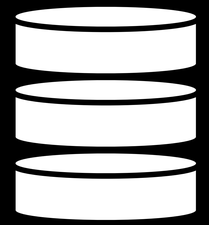
```
// Initialize the decryption key
let key = SymmetricKey(data: keyData)

// Initialize the sealed box
guard let sealedBox = ChaChaPoly.SealedBox(combined: downloadedData) else {
    throw MapDownloaderError.invalidDownload
}

// Open the sealed box (authenticates + decrypts)
let mapData = try ChaChaPoly.open(sealedBox, using: key)
```

Signatures

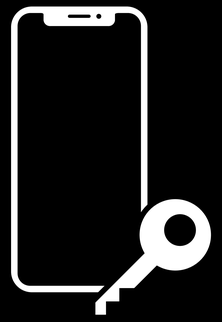
Authenticates data using a private key

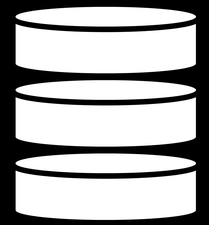
Verifies data using the associated public key

Using a Signature to Authorize an Operation

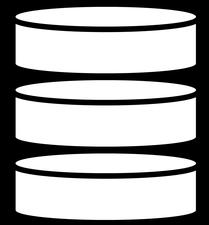


Using a Signature to Authorize an Operation

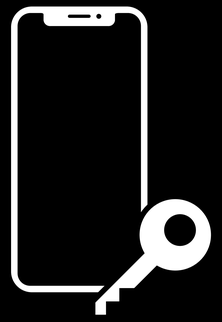




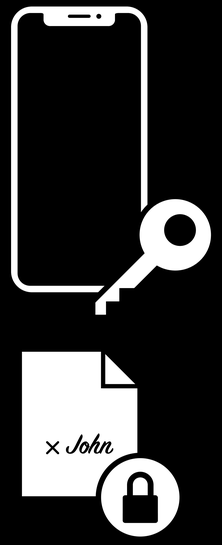
Using a Signature to Authorize an Operation



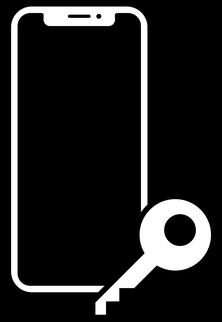
Using a Signature to Authorize an Operation

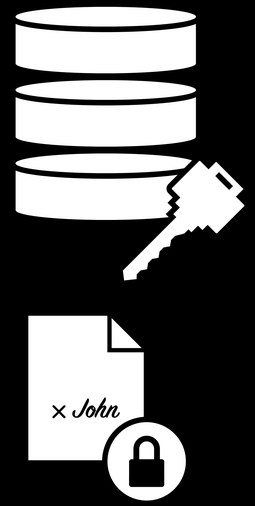


Using a Signature to Authorize an Operation

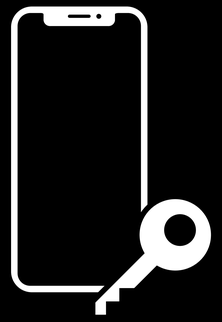


Using a Signature to Authorize an Operation

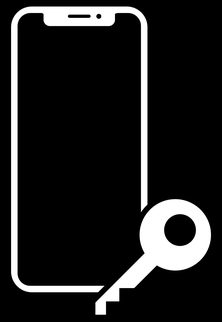




Using a Signature to Authorize an Operation



Using a Signature to Authorize an Operation



Using a Signature to Authorize an Operation

Using a Signature to Authorize an Operation

```
// Generate private key and register public key with server
let privateKey = P256.Signing.PrivateKey()
let publicKeyData = privateKey.publicKey.compactRepresentation!

// Store privateKey in Keychain
...
```

Using a Signature to Authorize an Operation

```
// Generate private key and register public key with server
let privateKey = P256.Signing.PrivateKey()
let publicKeyData = privateKey.publicKey.compactRepresentation!

// Store privateKey in Keychain
...


// Signing content
let signature = try privateKey.signature(for: transactionData)
```

Apple CryptoKit

Swift Framework

Secure Algorithms

Apple CryptoKit

Swift Framework

Secure Algorithms

Secure Enclave

Secure Enclave

Secure Enclave

Using the Secure Enclave

Using the Secure Enclave

```
// Check that the device has a Secure Enclave
if !SecureEnclave.isAvailable {
    // Handle devices without Secure Enclave
}
```

Using the Secure Enclave

```
// Check that the device has a Secure Enclave
if !SecureEnclave.isAvailable {
    // Handle devices without Secure Enclave
}

// Generate private key and register public key with server
let privateKey = try SecureEnclave.P256.Signing.PrivateKey()
let publicKeyData = privateKey.publicKey.compactRepresentation!

// Store privateKey in Keychain
...

// Producing a signature
let signature = try privateKey.signature(for: transactionData)
```

Constraining Key Usage

```
let accessControl = SecAccessControlCreateWithFlags(nil,
                                                    kSecAttrAccessibleWhenUnlockedThisDeviceOnly,
                                                    [.privateKeyUsage, .userPresence],
                                                    nil)!

let privateKey = try SecureEnclave.P256.Signing.PrivateKey(accessControl: accessControl)
```

Constraining Key Usage

```
let accessControl = SecAccessControlCreateWithFlags(nil,
                                                    kSecAttrAccessibleWhenUnlockedThisDeviceOnly,
                                                    [.privateKeyUsage, .userPresence],
                                                    nil)!

let privateKey = try SecureEnclave.P256.Signing.PrivateKey(accessControl: accessControl)
```

Constraining Key Usage

```
let accessControl = SecAccessControlCreateWithFlags(nil,
                                                    kSecAttrAccessibleWhenUnlockedThisDeviceOnly,
                                                    [.privateKeyUsage, .userPresence],
                                                    nil)!

let privateKey = try SecureEnclave.P256.Signing.PrivateKey(accessControl: accessControl)
```

Constraining Key Usage

```
let accessControl = SecAccessControlCreateWithFlags(nil,
                                                    kSecAttrAccessibleWhenUnlockedThisDeviceOnly,
                                                    [.privateKeyUsage, .userPresence],
                                                    nil)!

let privateKey = try SecureEnclave.P256.Signing.PrivateKey(accessControl: accessControl)
```

Constraining Key Usage

```
let accessControl = SecAccessControlCreateWithFlags(nil,
                                                    kSecAttrAccessibleWhenUnlockedThisDeviceOnly,
                                                    [.privateKeyUsage, .userPresence],
                                                    nil)!

let privateKey = try SecureEnclave.P256.Signing.PrivateKey(accessControl: accessControl)
```

Constraining Key Usage

```
let accessControl = SecAccessControlCreateWithFlags(nil,
                                                    kSecAttrAccessibleWhenUnlockedThisDeviceOnly,
                                                    [.privateKeyUsage, .userPresence],
                                                    nil)!

let privateKey = try SecureEnclave.P256.Signing.PrivateKey(accessControl: accessControl)
```

Customizing Authentication Context

```
let accessControl = SecAccessControlCreateWithFlags(nil,
                                                    kSecAttrAccessibleWhenUnlockedThisDeviceOnly,
                                                    [.privateKeyUsage, .userPresence],
                                                    nil)!

let authContext = LAContext()
authContext.touchIDAuthenticationAllowableReuseDuration = 10
authContext.localizedReason = "Authorizing $10 transfer to Bob."

let privateKey = try SecureEnclave.P256.Signing.PrivateKey(accessControl: accessControl
                                                           authenticationContext: authContext)
```

Customizing Authentication Context

```
let accessControl = SecAccessControlCreateWithFlags(nil,
                                                    kSecAttrAccessibleWhenUnlockedThisDeviceOnly,
                                                    [.privateKeyUsage, .userPresence],
                                                    nil)!

let authContext = LAContext()
authContext.touchIDAuthenticationAllowableReuseDuration = 10
authContext.localizedReason = "Authorizing $10 transfer to Bob."

let privateKey = try SecureEnclave.P256.Signing.PrivateKey(accessControl: accessControl
                                                           authenticationContext: authContext)
```

Customizing Authentication Context

```
let accessControl = SecAccessControlCreateWithFlags(nil,
                                                    kSecAttrAccessibleWhenUnlockedThisDeviceOnly,
                                                    [.privateKeyUsage, .userPresence],
                                                    nil)!

let authContext = LAContext()
authContext.touchIDAuthenticationAllowableReuseDuration = 10
authContext.localizedReason = "Authorizing $10 transfer to Bob."

let privateKey = try SecureEnclave.P256.Signing.PrivateKey(accessControl: accessControl
                                                           authenticationContext: authContext)
```

Customizing Authentication Context

```
let accessControl = SecAccessControlCreateWithFlags(nil,
                                                    kSecAttrAccessibleWhenUnlockedThisDeviceOnly,
                                                    [.privateKeyUsage, .userPresence],
                                                    nil)!

let authContext = LAContext()
authContext.touchIDAuthenticationAllowableReuseDuration = 10
authContext.localizedReason = "Authorizing $10 transfer to Bob."

let privateKey = try SecureEnclave.P256.Signing.PrivateKey(accessControl: accessControl
                                                           authenticationContext: authContext)
```

Apple CryptoKit

Swift Framework

Secure Algorithms

Secure Enclave

Apple CryptoKit

Swift Framework

Secure Algorithms

Secure Enclave

Performance

Performance

Performance

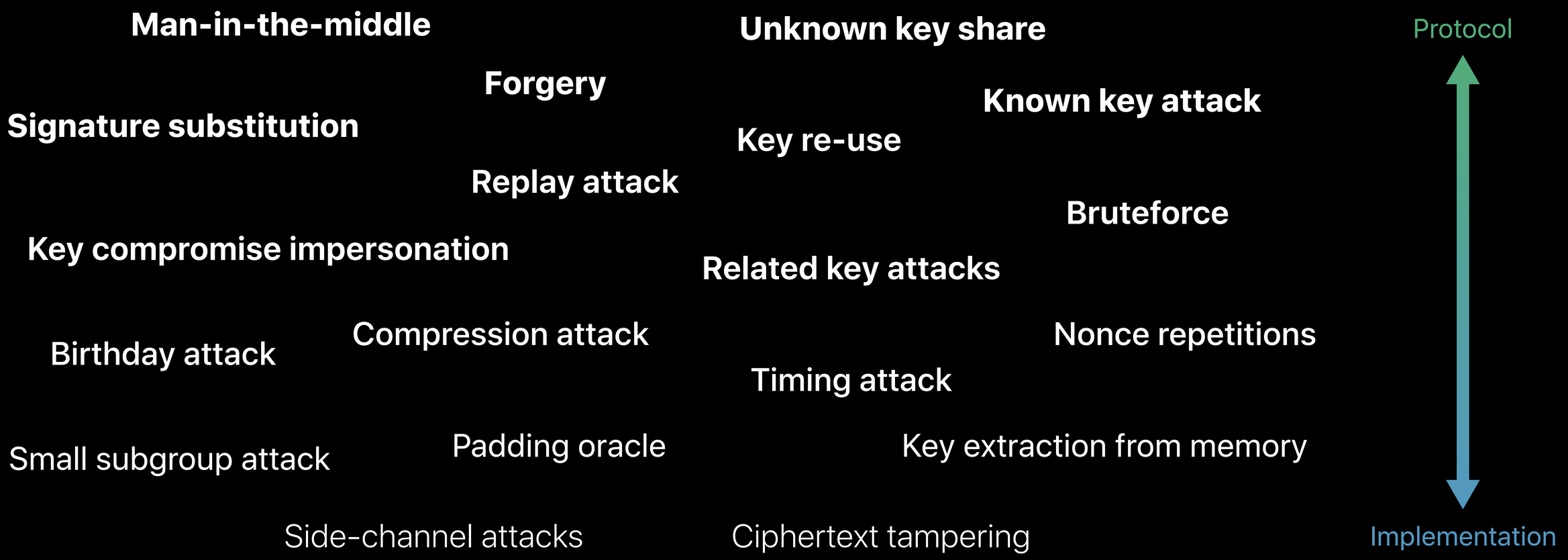
Built on top of corecrypto

Performance

Built on top of corecrypto

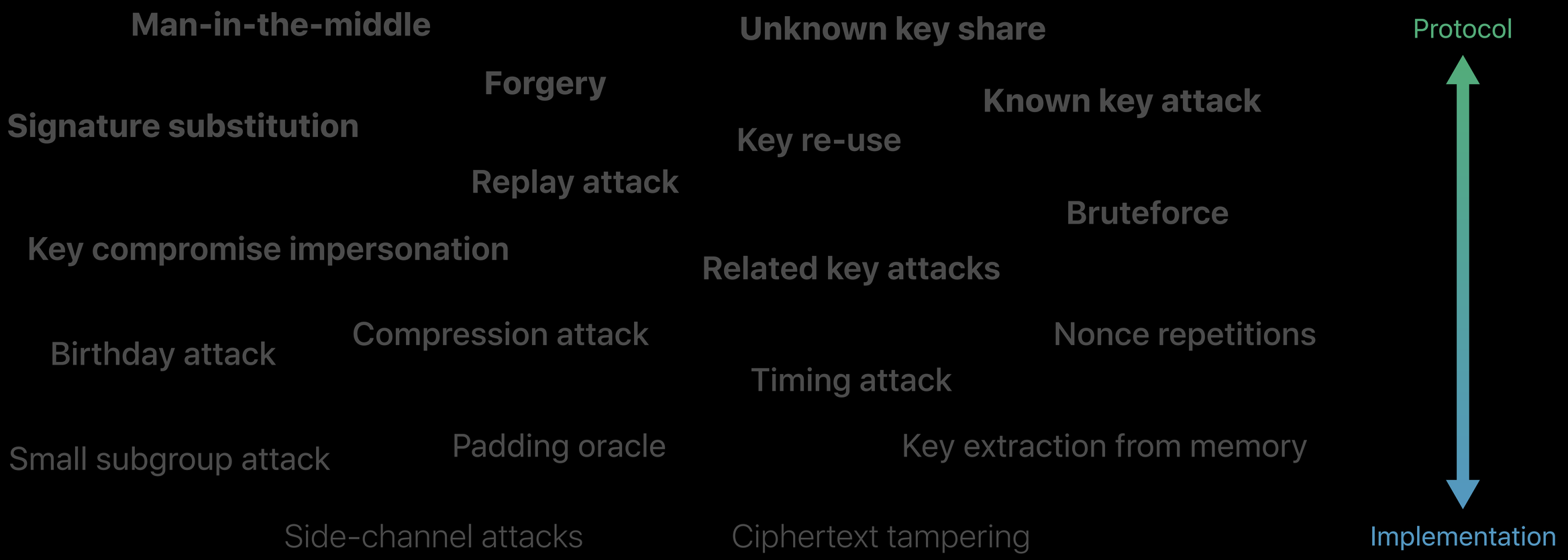
Hand-tuned assembly code

Cryptography Is Hard to Get Right



Cryptography Is Getting Easier to Get Right

Cryptography Is Getting Easier to Get Right



More Information

developer.apple.com/wwdc19/709

Security Lab

Thursday, 2:00


WWDC19