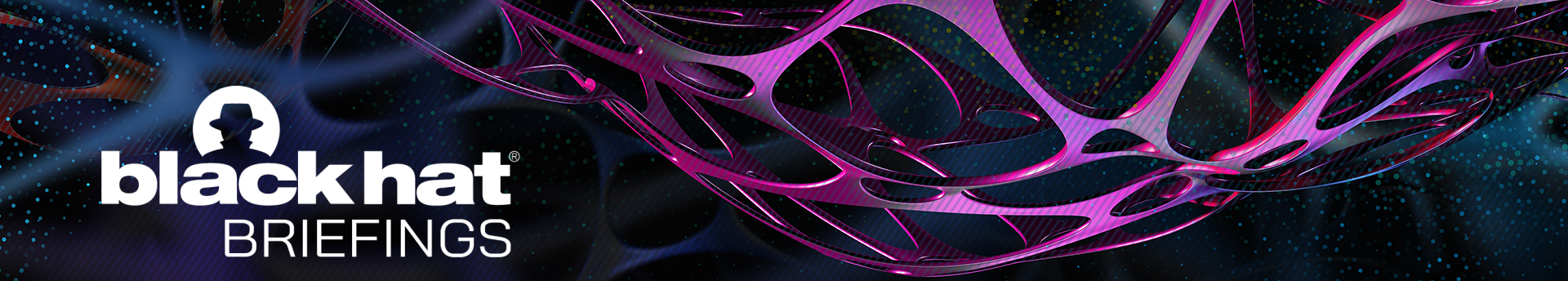


Decoding Signal: Understanding the Real Privacy Guarantees of E2EE

Ibrahim M. ElSayed

Agenda

- Setting the scene
- Attack surface
- 1:1 Messages
- Linked devices
- Conclusion

$whoami

- Ibrahim M. ElSayed (@the_st0rm)
- Security Engineer
  - Meta
  - Signal
  - Lacework
- Focus on Static Analysis
- Messaging application enthuthiast
  - Whatsapp - 2018
  - NSO attacks

Disclaimer

- Opinions shared are my own, not my employer
- The focus is purely technical
- Any app comparisons made are focused only technology-based and do not reference specific products by name

What to expect?

- A security review (I’m not a crypto expert)
- Close collaboration with the Signal team
- Focus on Signal 1:1 Messaging (no groups/calls)
- Takeaways: how signal works, privacy guarantees and vulnz (all fixed)

Methodology

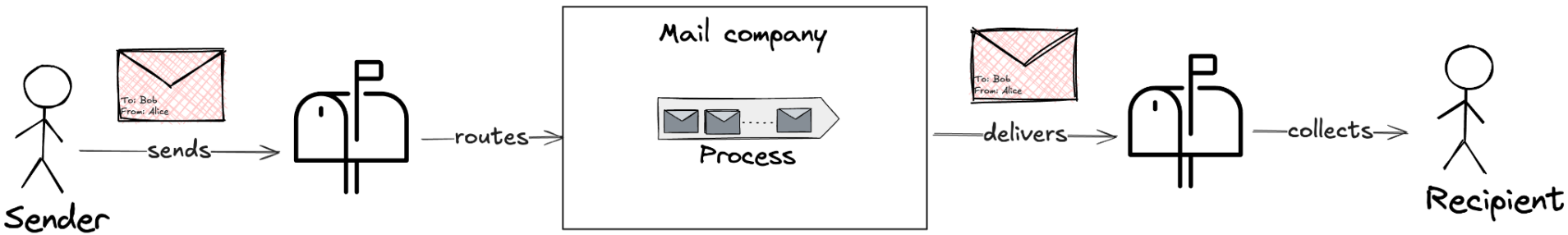
- Design: What the system is supposed to do
- Intent: What the engineer understood
- Implementation: The actual code that was written.
- Execution:How the code behaves in practice

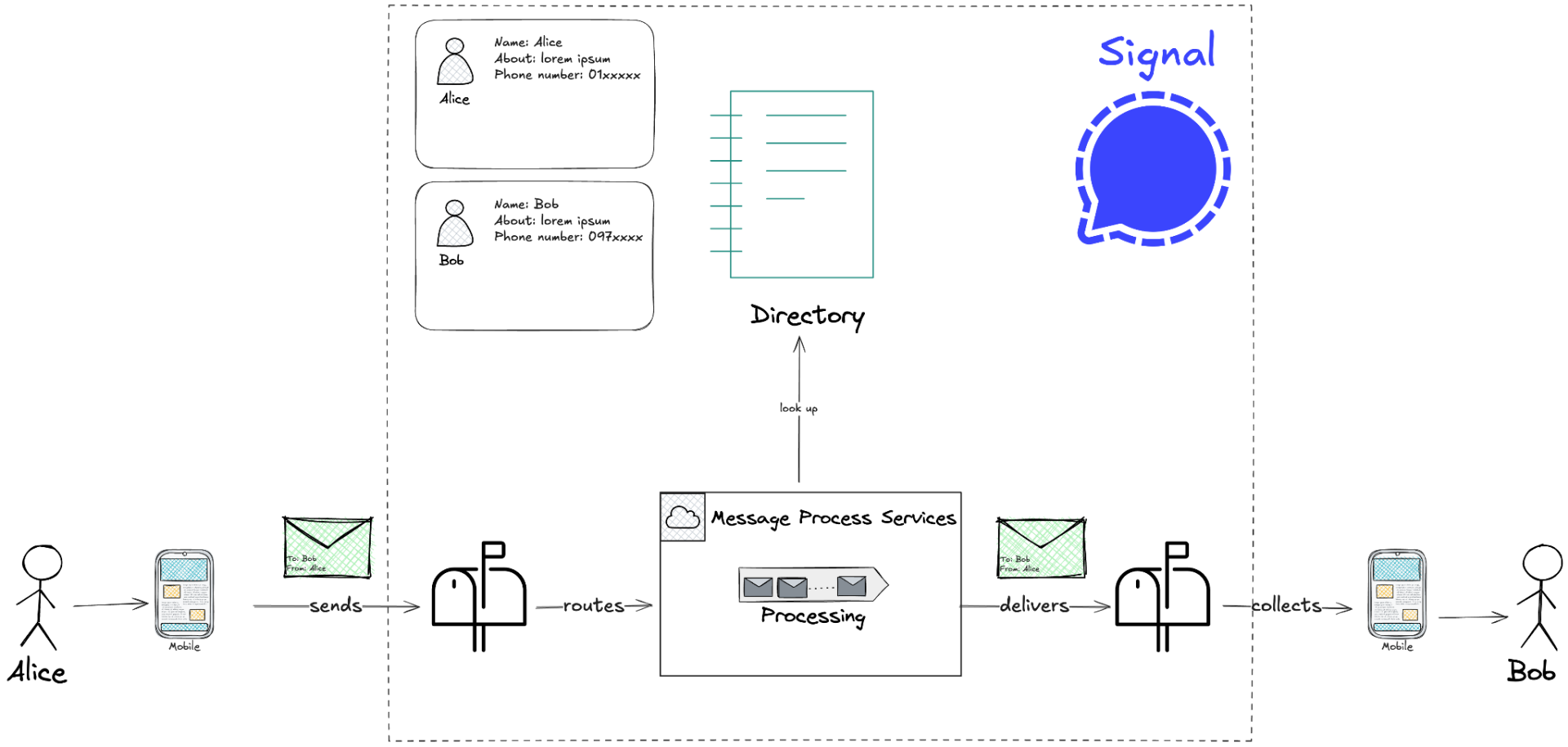
Methodology - Vulnerability classes

- Language-specific: Memory corruption in C++
- Application-specific: SQL injection
- Logic-based: Broken authorization
- Product-specific: Unique to the app's domain e.g., leaking if 2 users are communicating

Understanding Signal's Architecture

Sending a mail




Name: Alice
About: lorem ipsum
Phone number: 01xxxxx
Alice
Name: Bob
About: lorem ipsum
Phone number: 097xxxx
Bob
Directory
Signal
look up
Message Process Services
Processing
Alice
Mobile
To: Bob
From: Alice
sends
routes
delivers
collects
Mobile
Bob

Attack Surface

- Backend Services
  - Mostly Java and Rust
- Clients
  - Library Rust
  - Android: Kotlin + Java
  - iOS: Swift + ObjC
  - Desktop: Electron App

Attack Surface

- Backend Services
  - Chat server: ~230K
  - Storage Server: ~40K
- Clients
  - Signal Library: ~100K LoC Rust
  - Android: ~300K
  - Desktop: ~300K
  - iOS: ~500K (~90% Swift)

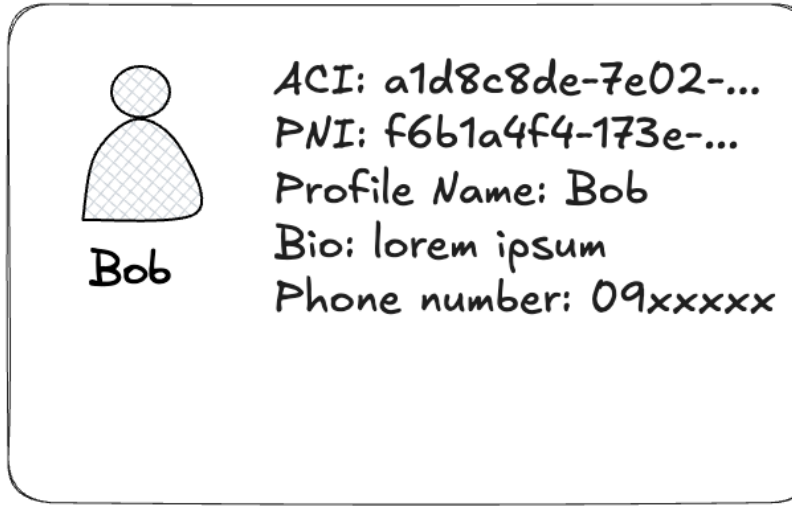
Attack Surface

- Expectation of E2EE applications
  - Server is malicious
  - Network is hostile
- Protecting
  - Message Content
  - Metadata
    - Profile
    - Messages

Profile Data

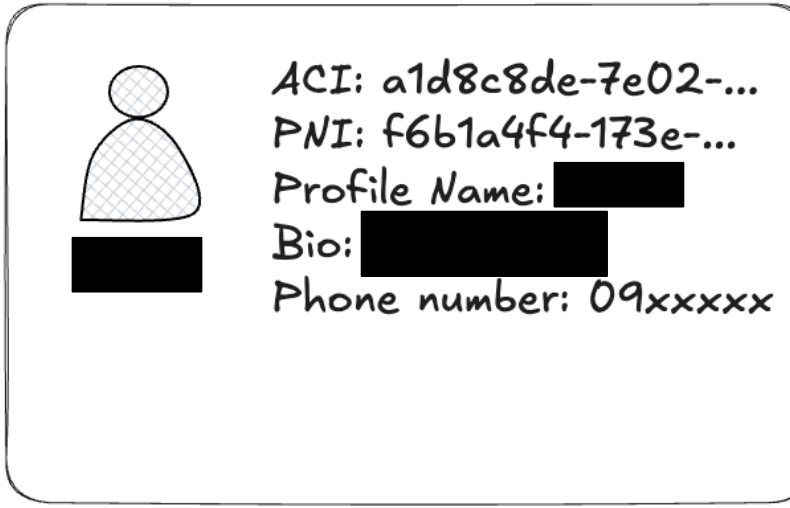
Profile Data

- Account identifiers
  - ACI: Account Identifier
  - PNI: Phone Number Identifier
- Profile data
  - Name, Avatar, Bio, Badges, Device caps, …



Profile Data

- Stored in Storage Service (Java)
- Encrypted in AES-256-GCM
- Key generation: Clients
- Key sharing: every message after conversation is accepted
- Key rotation: Blocking a user



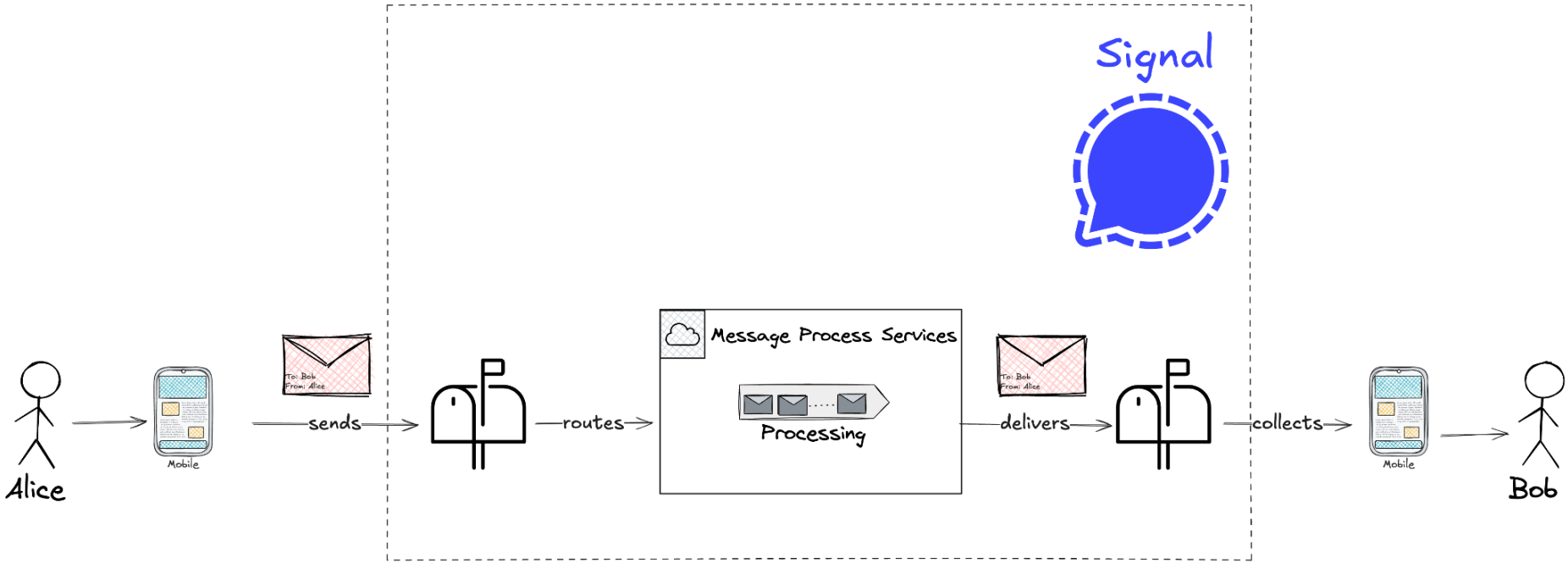
Profile Data Storage

Signal

ACI	name	bio	image_url
a1d8c8de..	E1Q9x/L5OGb8v6mZq3KcHg==	XaB2+0fjTwXVoMJhVXJPZg==	aqn4Z2s3ZnlmbHNqa3g0OA==
91ad7c55..	ZTlwT2xheHh0eGl2ZDIzYw==	YT1lbnU4KkJxd0g9Q2Zpbk4=	NkJubGlyZUdGejN0YlN1Yw==

user_id	name	bio	image_url
839201	Alice W.	"Living the dream 🌍"	https://cdn.random.net/u1.png
283932	Bob R.	"Coffee ☕ > Everything"	https://cdn.mockcdn.io/bob.jpg

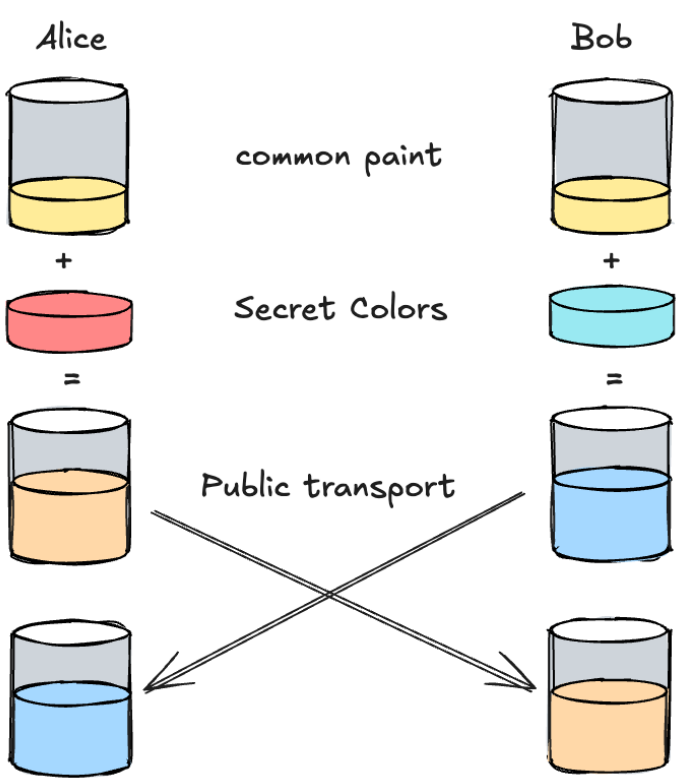
Messages


Signal
Alice
Mobile
To: Bob
From: Alice
sends
routes
Message Process Services
Processing
delivers
To: Bob
From: Alice
collects
Mobile
Bob

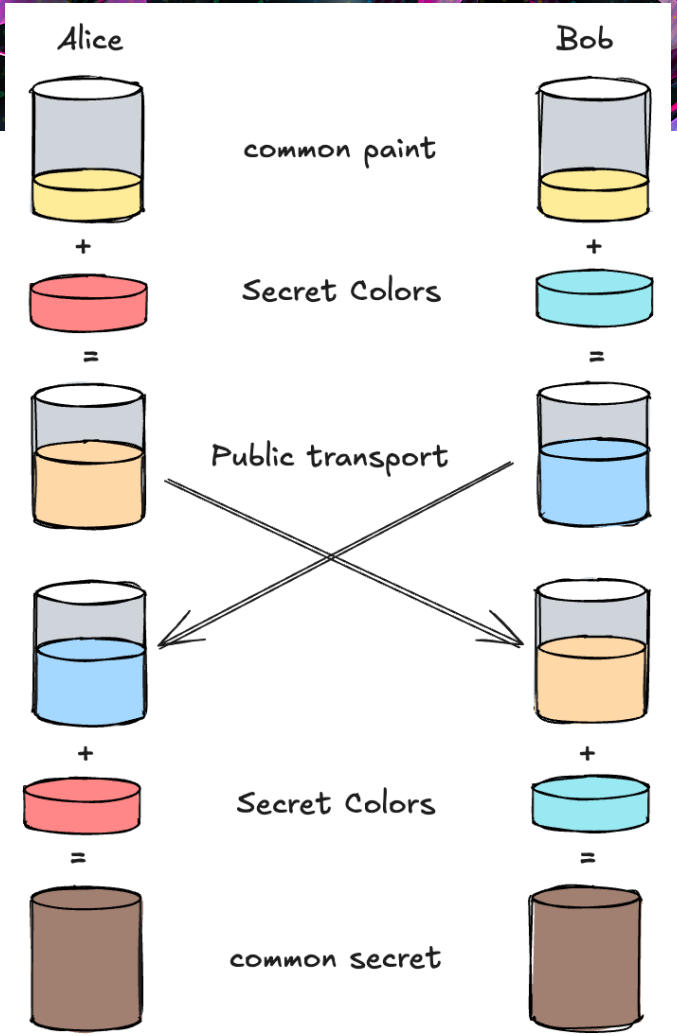
Diffie-Hellman

Diffie-Hellman

Diffie-Hellman

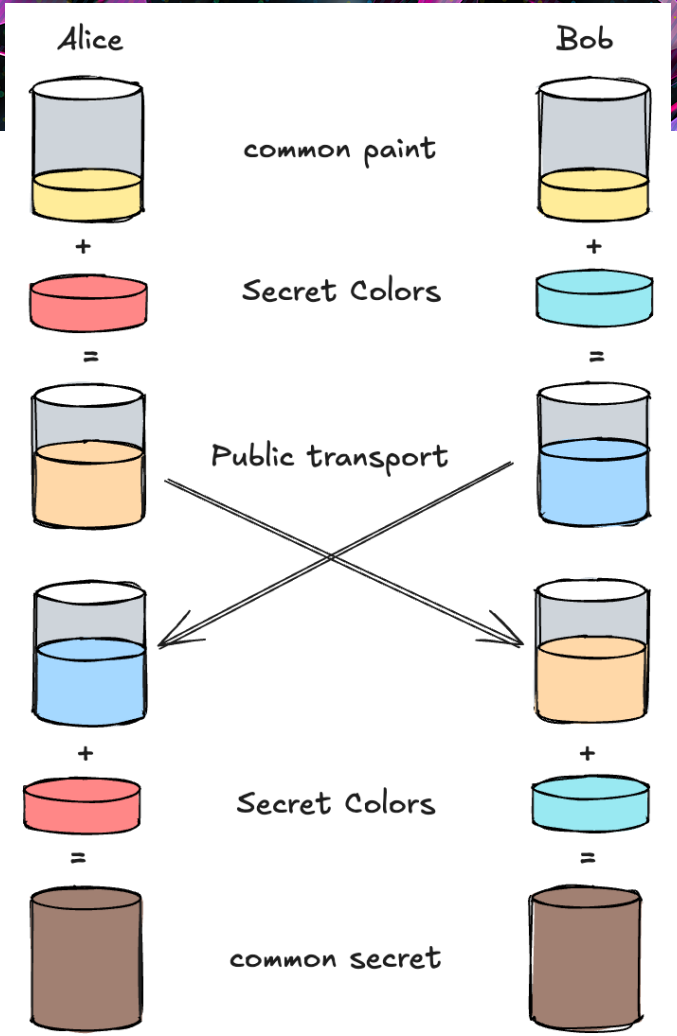


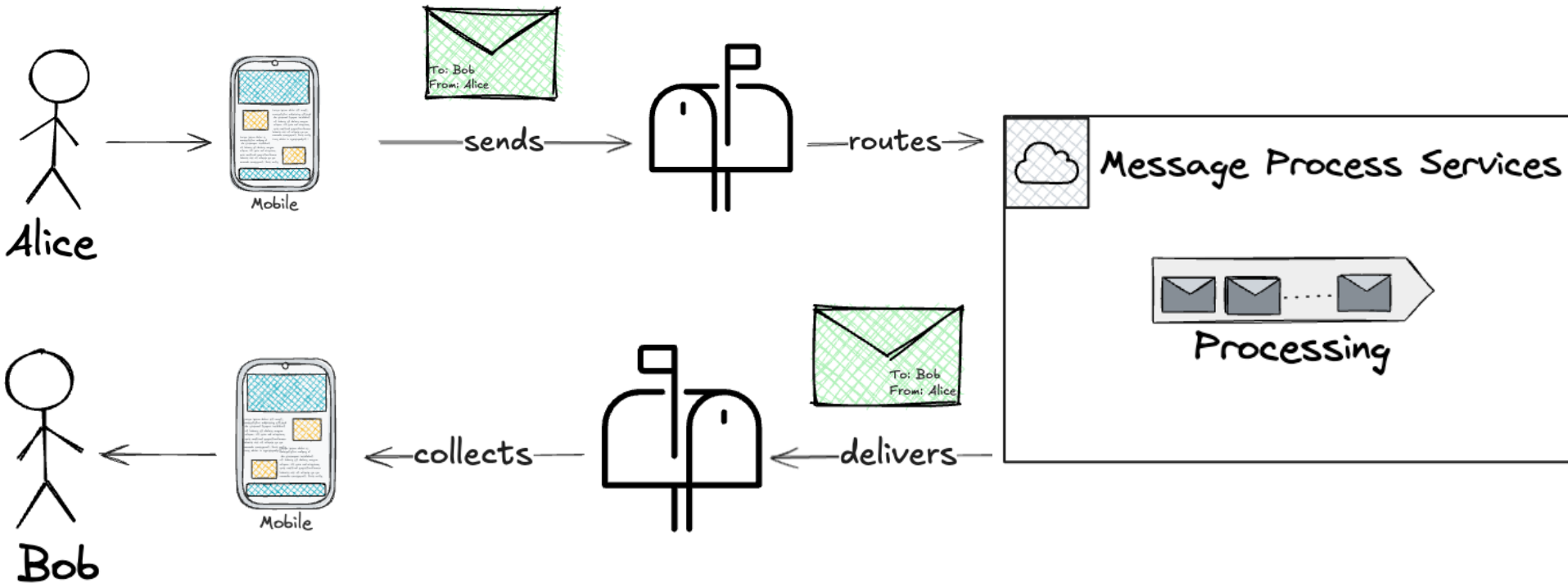
Diffie-Hellman



Diffie-Hellman

- EC-DH
- Pre-keys are signed by Identity keys
- Identity keys are the most critical keys for Signal account




Alice
Mobile
To: Bob
From: Alice
sends
routes
Message Process Services
Processing
To: Bob
From: Alice
delivers
collects
Mobile
Bob

Backward Secrecy

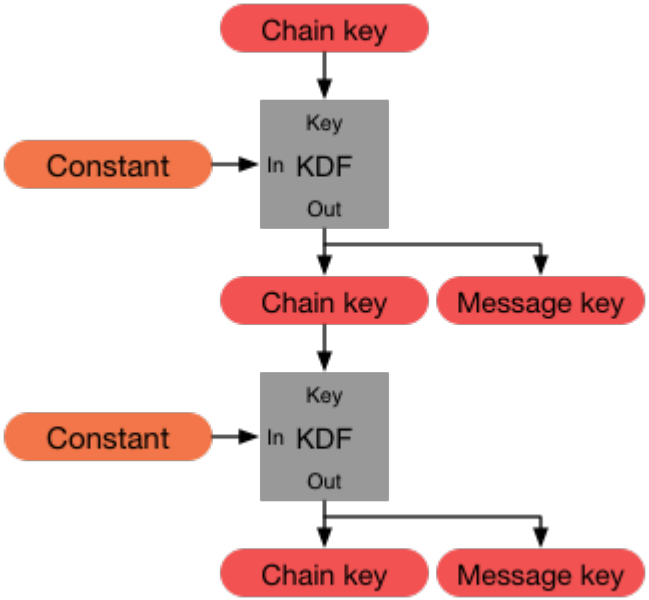
- Compromised device shouldn’t compromise previous conversations

Backward Secrecy

- Compromised device shouldn’t compromise previous conversations
- Design: Every message is encrypted with a unique key
- Symmetric key ratchet

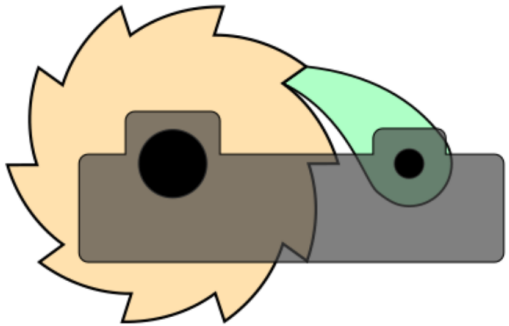
Backward Secrecy

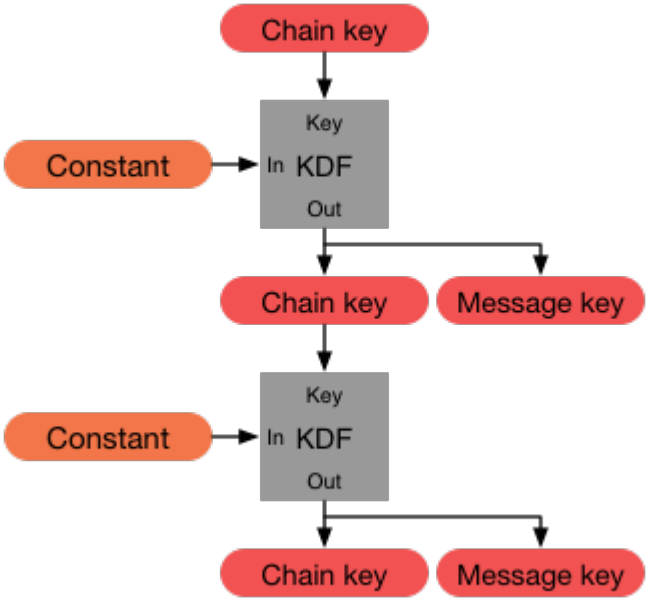
- Chain key from DH agreement
- Message key for encryption
- Chain key to derive next message Key
- Intermediate chain-keys are deleted



Backward Secrecy

- Chain key from DH agreement
- Message key for encryption
- Chain key to derive next message Key
- Intermediate chain-keys are deleted





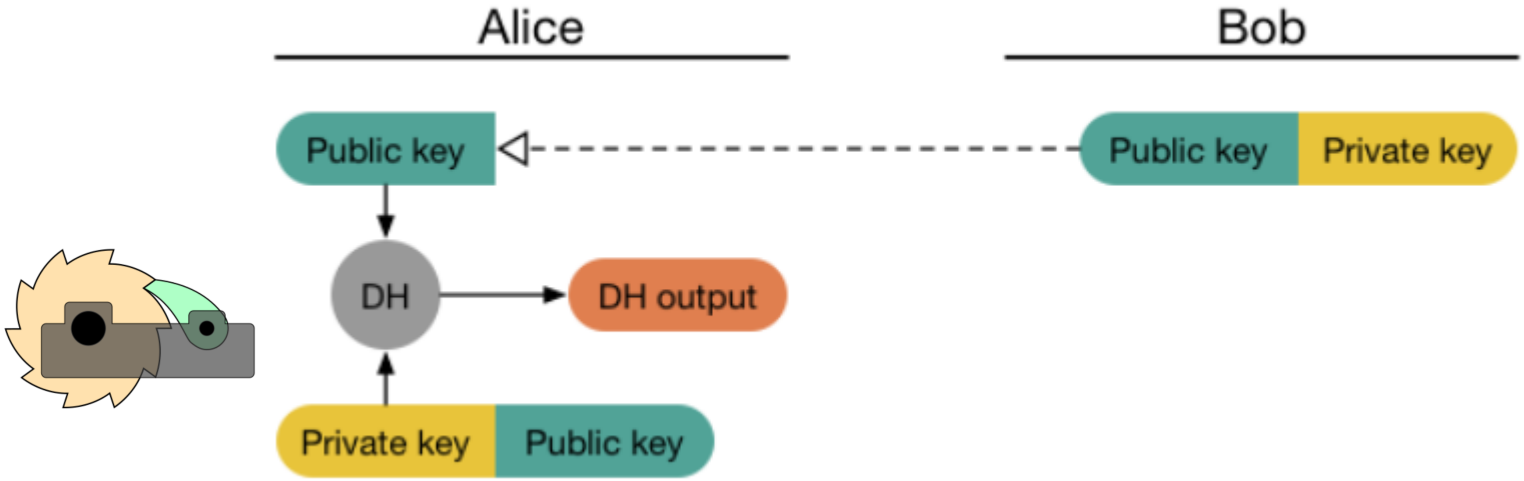
Future Secrecy

- Compromised key shouldn’t compromise future conversations

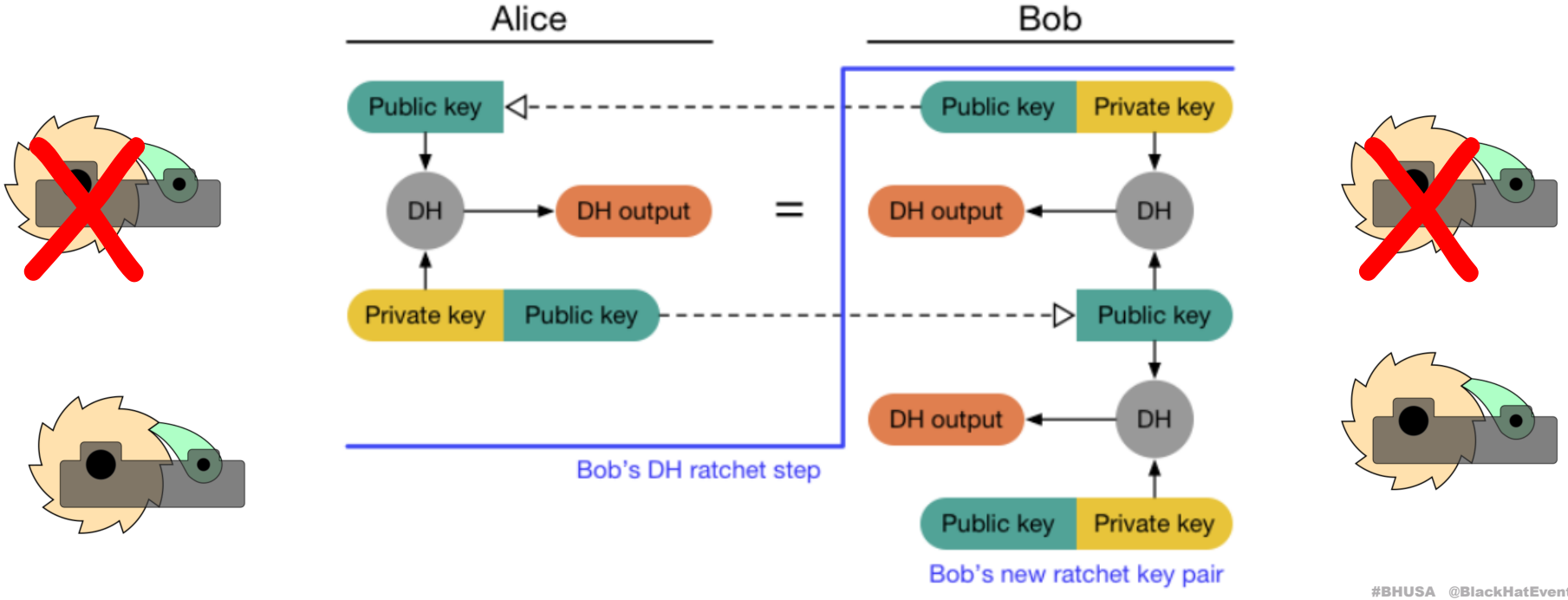
Future Secrecy

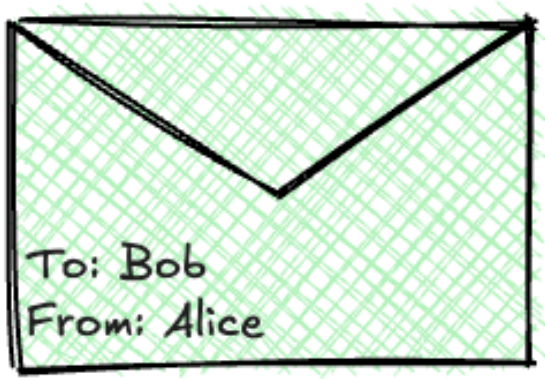
- Compromised key shouldn’t compromise future conversations
- Design: Assume the secret channel is compromised. Establish Secret keys on an unsafe channel
- Diffie-Hellman Ratchet

Diffie-Hellman ratchet

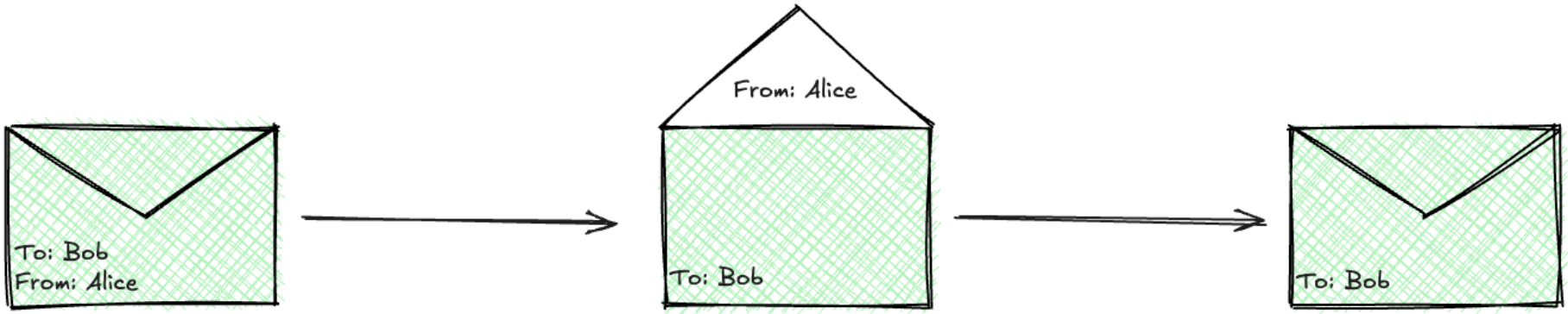


Diffie-Hellman ratchet




To: Bob
From: Alice

Unidentified Sender (UD Sender)



Technology check

	Signal	App1	App2	App3	App4
Encrypted Profile	✅	❌	❌	❌	❌
Encrypted Messages	✅	❌ (optional)	✅	✅	✅
Single ratchet	✅	✅	✅	✅	✅
double ratchet	✅	❌	❌	✅	❌
Sealed Sender	✅	❌	❌	❌	❌

Implementation

```
 0 message Envelope {
 1   optional Type   type                = 1;
 2   reserved      /*sourceE164*/          2;
 3   optional string sourceServiceId     = 11;
 4   optional uint32 sourceDevice        = 7;
 5   optional string destinationServiceId = 13;
 6   reserved      /*relay*/               3;
 7   optional uint64 timestamp           = 5;
 8   reserved      /*legacyMessage*/       6;
 9   optional bytes  content             = 8;
10   optional string serverGuid          = 9;
11   optional uint64 serverTimestamp     = 10;
12   optional bool   urgent              = 14
13   optional bool   story               = 16;
14   optional bytes  reportingToken      = 17;
15 }
```

Contains encrypted Content

```
0 message Envelope {
1   enum Type {
2     UNKNOWN                = 0;
3     CIPHERTEXT             = 1;
4     KEY_EXCHANGE           = 2;
5     PREKEY_BUNDLE          = 3;
6     RECEIPT                = 5;
7     UNIDENTIFIED_SENDER    = 6;
8     reserved 7;
9     PLAINTEXT_CONTENT      = 8;
10  }
```

Plaintext Envelopes

Design

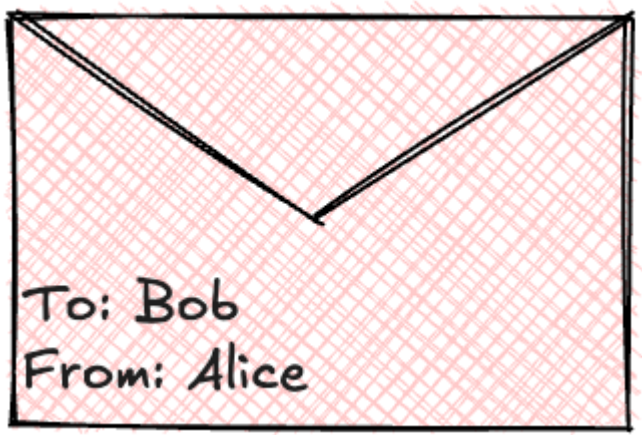
When the recipient device receives an undecryptable message, the recipient device sends an unencrypted **retry request** message to the original sending device's mailbox, containing the undecryptable message's *MessageID*.

When the original sending device fetches a retry request along with the relevant *UserID* and *DeviceID* of the device that sent the retry request, the original sending device executes the following **resending** process:

https://signal.org/docs/specifications/sesame/

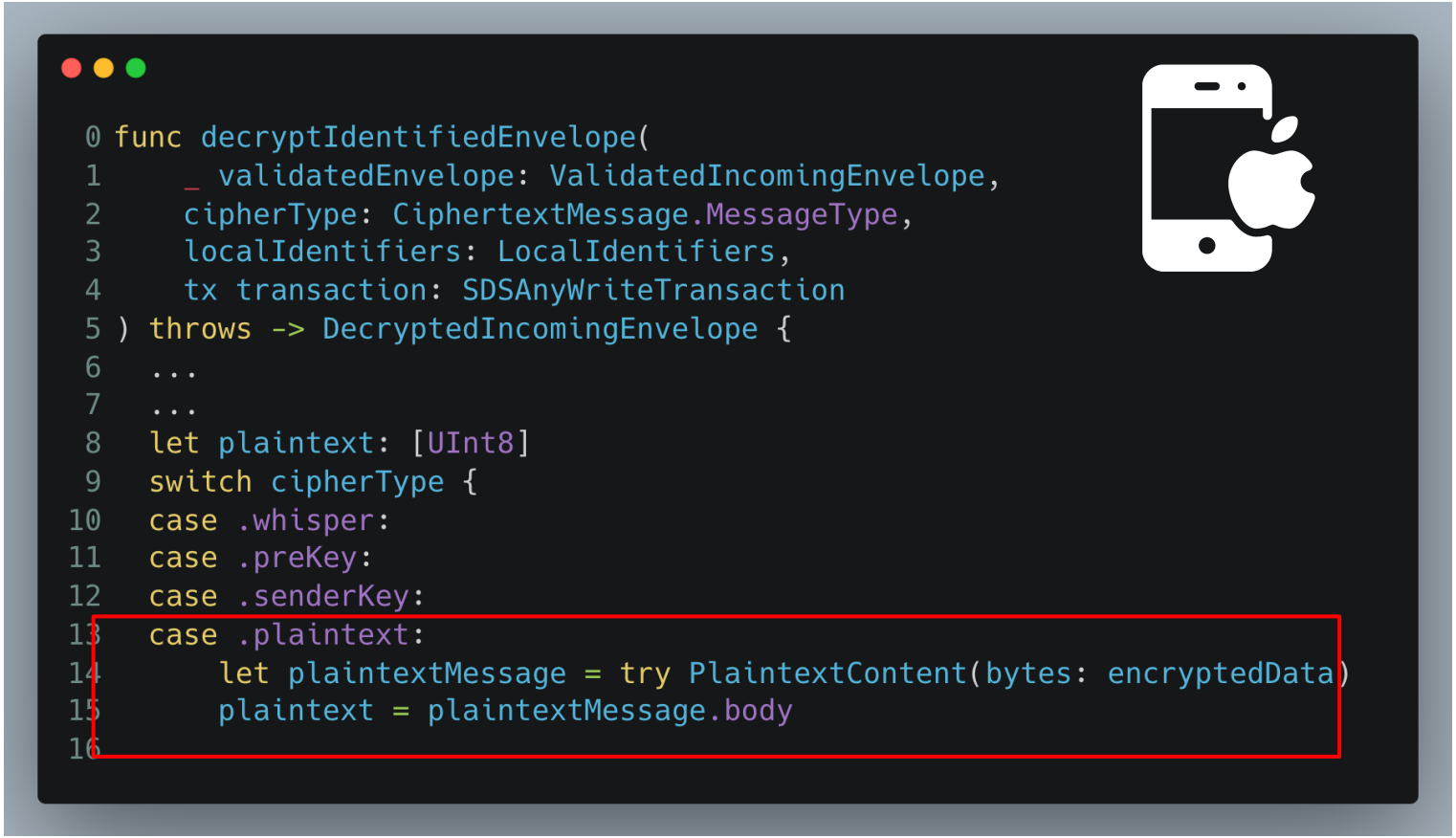
Plaintext Envelopes

Implementation



```
0 func decryptIdentifiedEnvelope(
1     _ validatedEnvelope: ValidatedIncomingEnvelope,
2     cipherType: CiphertextMessage.MessageType,
3     localIdentifiers: LocalIdentifiers,
4     tx transaction: SDSAnyWriteTransaction
5 ) throws -> DecryptedIncomingEnvelope {
6   ...
7   ...
8   let plaintext: [UInt8]
9   switch cipherType {
10    case .whisper:
11    case .preKey:
12    case .senderKey:
13    case .plaintext:
14
```

```swift
func decryptIdentifiedEnvelope(
    _ validatedEnvelope: ValidatedIncomingEnvelope,
    cipherType: CiphertextMessage.MessageType,
    localIdentifiers: LocalIdentifiers,
    tx transaction: SDSAnyWriteTransaction
) throws -> DecryptedIncomingEnvelope {
  ...
  ...
  let plaintext: [UInt8]
  switch cipherType {
  case .whisper:
      let message = try SignalMessage(bytes: encryptedData)
      plaintext = try signalDecrypt(
          message: message,
          from: protocolAddress,
          sessionStore: signalProtocolStore.sessionStore,
          identityStore: identityManager.libSignalStore(for: localIdentity, tx: transaction.asV2Write),
          context: transaction
      )
      sendReactiveProfileKeyIfNecessary(to: sourceAci, tx: transaction)
  case .preKey:
  case .senderKey:
  case .plaintext:
```



```
 0 func decryptIdentifiedEnvelope(
 1     _ validatedEnvelope: ValidatedIncomingEnvelope,
 2     cipherType: CiphertextMessage.MessageType,
 3     localIdentifiers: LocalIdentifiers,
 4     tx transaction: SDSAnyWriteTransaction
 5 ) throws -> DecryptedIncomingEnvelope {
 6   ...
 7   ...
 8   let plaintext: [UInt8]
 9   switch cipherType {
10   case .whisper:
11   case .preKey:
12   case .senderKey:
13   case .plaintext:
14       let plaintextMessage = try PlaintextContent(bytes: encryptedData)
15       plaintext = plaintextMessage.body
16
```

```
 0 message Content {
 1   optional DataMessage         dataMessage                = 1;
 2   optional SyncMessage         syncMessage                = 2;
 3   optional CallMessage         callMessage                = 3;
 4   optional NullMessage         nullMessage                = 4;
 5   optional ReceiptMessage      receiptMessage             = 5;
 6   optional TypingMessage       typingMessage              = 6;
 7   optional bytes               senderKeyDistributionMessage = 7;
 8   optional bytes               decryptionErrorMessage     = 8;
 9   optional StoryMessage        storyMessage               = 9;
10   optional PniSignatureMessage pniSignatureMessage        = 10;
11   optional EditMessage         editMessage                = 11;
12 }
```

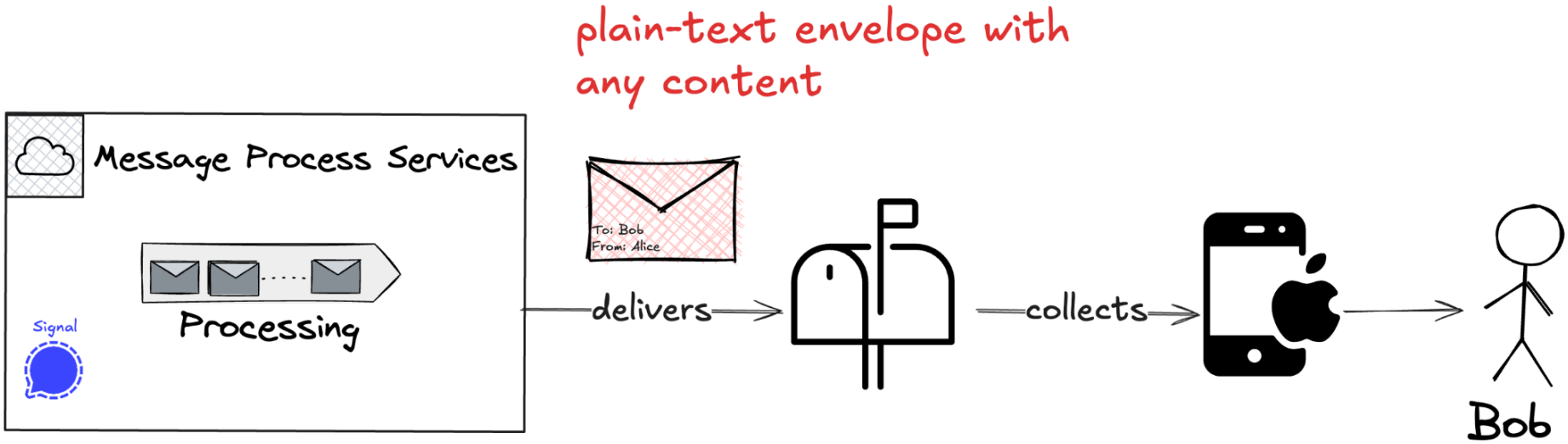
```
0 func handleRequest(
1     _ request: MessageReceiverRequest,
2     context: DeliveryReceiptContext,
3     localIdentifiers: LocalIdentifiers,
4     tx: SDSAnyWriteTransaction
5 ) {
6     let protoContent = request.protoContent
7
8     switch request.messageType {
9     case .syncMessage(let syncMessage):
10    case .dataMessage(let dataMessage):
11    case .callMessage(let callMessage):
12    case .typingMessage(let typingMessage):
13    case .nullMessage:
14    case .receiptMessage(let receiptMessage):
15    case .decryptionErrorMessage(let decryptionErrorMessage):
16    case .storyMessage(let storyMessage):
17    case .editMessage(let editMessage):
18    case .handledElsewhere:
19    case .none:
20        Logger.warn("Ignoring envelope with unknown type.")
21    }
22 }
```

```
0 func handleRequest(
1     _ request: MessageReceiverRequest,
2     context: DeliveryReceiptContext,
3     localIdentifiers: LocalIdentifiers,
4     tx: SDSAnyWriteTransaction
5 ) {
6     let protoContent = request.protoContent
7
8     switch request.messageType {
9     case .syncMessage(let syncMessage):
10    case .dataMessage(let dataMessage):
11      handleIncomingEnvelope(request: request, dataMessage:
  dataMessage, localIdentifiers: localIdentifiers, tx: tx)
12    case .callMessage(let callMessage):
13    case .typingMessage(let typingMessage):
14    case .nullMessage:
15    case .receiptMessage(let receiptMessage):
16    case .decryptionErrorMessage(let decryptionErrorMessage):
17    case .storyMessage(let storyMessage):
18    case .editMessage(let editMessage):
19    case .handledElsewhere:
20    case .none:
21        Logger.warn("Ignoring envelope with unknown type.")
22    }
23 }
```

Plain-Text envelope vulnerability

- iOS Clients process plain-text envelopes
  - ex: DataMessage, Edit Message
- Requires a malicious server

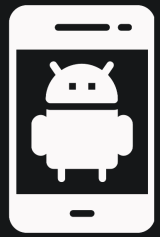
Impact


plain-text envelope with any content
Message Process Services
Processing
Signal
To: Bob
From: Alice
delivers
collects
Bob

```
let hasDecryptionError = (
        content?.decryptionErrorMessage != nil
    )
    let hasAnythingElse = (
        content?.dataMessage != nil
        || content?.syncMessage != nil
        || content?.callMessage != nil
        || content?.nullMessage != nil
        || content?.receiptMessage != nil
        || content?.typingMessage != nil
        || content?.storyMessage != nil
        || content?.pniSignatureMessage != nil
        || content?.senderKeyDistributionMessage != nil
        || content?.unknownFields != nil
    )
    if hasDecryptionError && hasAnythingElse {
        throw OWSGenericError("Message content must contain one type.")
    }
    if let isPlaintextCipher, isPlaintextCipher != hasDecryptionError {
        throw OWSGenericError("Plaintext ciphers must have decryption errors.")
    }
}
```

https://github.com/signalapp/Signal-iOS/commit/6bdf97ae88a72e2c7e843d7e4937647fcfbbf339#diff-79ca5c51f765f3b6fc3d184a8fb115d7b21bbbdbb1a5eec225cd33d914fd2f5d

```
0 private Plaintext decryptInternal(
1    Envelope envelope,
2    long serverDeliveredTimestamp) {
3   if (envelope.type == Envelope.Type.PREKEY_BUNDLE) {
4   } else if (envelope.type == Envelope.Type.CIPHERTEXT) {
5   } else if (envelope.type == Envelope.Type.PLAINTEXT_CONTENT) {
6   } else if (envelope.type == Envelope.Type.UNIDENTIFIED_SENDER) {
7   } else {
8     throw new InvalidMetadataMessageException("Unknown type: " +
  envelope.type);
9   }
```



```
0 private Plaintext decryptInternal(Envelope envelope, long serverDeliveredTimestamp) {
1   if (envelope.type == Envelope.Type.PREKEY_BUNDLE) {
2   } else if (envelope.type == Envelope.Type.CIPHERTEXT) {
3     SignalProtocolAddress sourceAddress = new SignalProtocolAddress(
4         envelope.sourceServiceId,
5         envelope.sourceDevice);
6     SignalSessionCipher sessionCipher = new SignalSessionCipher(
7         sessionLock,
8         new SessionCipher(signalProtocolStore,
9         sourceAddress));
10    paddedMessage = sessionCipher.decrypt(new SignalMessage(envelope.content.toByteArray()));
11    metadata = new SignalServiceMetadata(
12        getSourceAddress(envelope),
13        envelope.sourceDevice,
14        envelope.timestamp,
15        envelope.serverTimestamp,
16        serverDeliveredTimestamp,
17        false,
18        envelope.serverGuid,
19        Optional.empty(),
20        envelope.destinationServiceId);
21  } else if (envelope.type == Envelope.Type.PLAINTEXT_CONTENT) {
22  } else if (envelope.type == Envelope.Type.UNIDENTIFIED_SENDER) {
23  } else {
24    throw new InvalidMetadataMessageException("Unknown type: " + envelope.type);
25  }
26
```

```
0 private Plaintext decryptInternal(Envelope envelope, long serverDeliveredTimestamp) {
1   if (envelope.type == Envelope.Type.PREKEY_BUNDLE) {
2   } else if (envelope.type == Envelope.Type.CIPHERTEXT) {
3   } else if (envelope.type == Envelope.Type.PLAINTEXT_CONTENT) {
4     paddedMessage = new PlaintextContent(envelope.content.toByteArray()).getBody();
5     metadata      = new SignalServiceMetadata(
6         getSourceAddress(envelope),
7         envelope.sourceDevice,
8         envelope.timestamp,
9         envelope.serverTimestamp,
10        serverDeliveredTimestamp,
11        false,
12        envelope.serverGuid,
13        Optional.empty(),
14        envelope.destinationServiceId);
15  } else if (envelope.type == Envelope.Type.UNIDENTIFIED_SENDER) {
16  } else {
17    throw new InvalidMetadataMessageException("Unknown type: " + envelope.type);
18  }
```

```kotlin
private fun createPlaintextResultIfInvalid(content: Content): Result? {
  val errors: MutableList<String> = mutableListOf()

  if (content.decryptionErrorMessage == null) {
    errors += "Missing DecryptionErrorMessage"
  }
  if (content.storyMessage != null) {
    errors += "Unexpected StoryMessage"
  }
  if (content.senderKeyDistributionMessage != null) {
    errors += "Unexpected SenderKeyDistributionMessage"
  }
  if (content.editMessage != null) {
    errors += "Unexpected EditMessage"
  }

  ...
  ...

  return if (errors.isNotEmpty()) {
    Result.Invalid("Invalid PLAINTEXT_CONTENT! Errors: $errors")
  } else {
    null
  }
}
```

DataMessage Check is missing!

```
0 private fun handleMessage(
1     senderRecipient: Recipient,
2     envelope: Envelope,
3     content: Content,
4     metadata: EnvelopeMetadata,
5     serverDeliveredTimestamp: Long,
6     processingEarlyContent: Boolean,
7     localMetric: SignalLocalMetrics.MessageReceive?
8 ) {
9     when {
10      content.dataMessage != null -> { ... }
11      content.syncMessage != null -> { ... }
12      content.callMessage != null -> { ... }
13      content.receiptMessage != null -> { ... }
14      content.typingMessage != null -> { ... }
15      content.storyMessage != null -> { ... }
16      content.decryptionErrorMessage != null -> { ... }
17      content.editMessage != null -> { ... }
18      content.senderKeyDistributionMessage != null
19        || content.pniSignatureMessage != null -> { ... }
20      else -> {
21        warn(envelope.timestamp!!, "Got unrecognized message!")
22      }
23    }
24 }
```

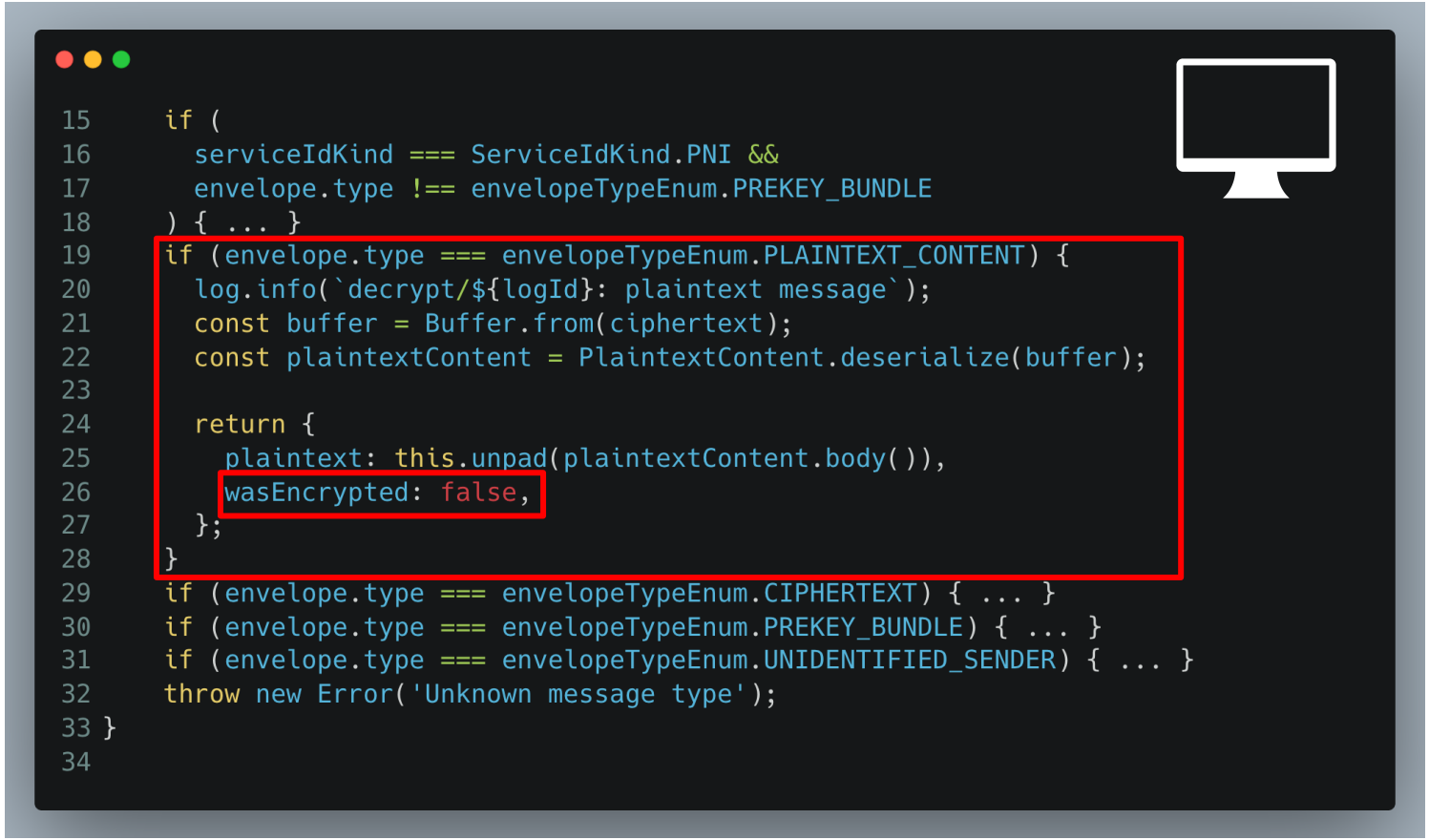
Plain-Text envelope vulnerability

- Android Clients process plain-text envelopes with DataMessage
- Required a malicious server

```
 0 private fun validatePlaintextContent(content: Content): Result?
   {
 1     val errors: MutableList<String> = mutableListOf()
 2
 3     if (content.decryptionErrorMessage == null) {
 4       errors += "Missing DecryptionErrorMessage"
 5     }
 6 +   if (content.dataMessage != null) {
 7 +     errors += "Unexpected DataMessage"
 8 +   }
 9     if (content.syncMessage != null) {
10       errors += "Unexpected SyncMessage"
11     }
12     if (content.callMessage != null) {
13       errors += "Unexpected CallMessage"
14     }
```

https://github.com/signalapp/Signal-Android/commit/8d8c21f2286d2449aaaf63669ea6e06e47ec076a

```
0 private async innerDecrypt(
1     stores: LockedStores,
2     envelope: UnsealedEnvelope,
3     ciphertext: Uint8Array,
4     serviceIdKind: ServiceIdKind
5 ): Promise<InnerDecryptResultType | undefined> {
6
7     const identifier = envelope.sourceServiceId;
8     const { sourceDevice } = envelope;
9     const { destinationServiceId } = envelope;
10    const address = new QualifiedAddress(
11      destinationServiceId,
12      Address.create(identifier, sourceDevice)
13    );
14
15    if (
16      serviceIdKind === ServiceIdKind.PNI &&
17      envelope.type !== envelopeTypeEnum.PREKEY_BUNDLE
18    ) { ... }
19    if (envelope.type === envelopeTypeEnum.PLAINTEXT_CONTENT) { ... }
20    if (envelope.type === envelopeTypeEnum.CIPHERTEXT) { ... }
21    if (envelope.type === envelopeTypeEnum.PREKEY_BUNDLE) { ... }
22    if (envelope.type === envelopeTypeEnum.UNIDENTIFIED_SENDER) { ... }
23    throw new Error('Unknown message type');
24 }
```



```
15      if (
16        serviceIdKind === ServiceIdKind.PNI &&
17        envelope.type !== envelopeTypeEnum.PREKEY_BUNDLE
18      ) { ... }
19      if (envelope.type === envelopeTypeEnum.PLAINTEXT_CONTENT) {
20        log.info(`decrypt/${logId}: plaintext message`);
21        const buffer = Buffer.from(ciphertext);
22        const plaintextContent = PlaintextContent.deserialize(buffer);
23
24        return {
25          plaintext: this.unpad(plaintextContent.body()),
26          wasEncrypted: false,
27        };
28      }
29      if (envelope.type === envelopeTypeEnum.CIPHERTEXT) { ... }
30      if (envelope.type === envelopeTypeEnum.PREKEY_BUNDLE) { ... }
31      if (envelope.type === envelopeTypeEnum.UNIDENTIFIED_SENDER) { ... }
32      throw new Error('Unknown message type');
33  }
34
```

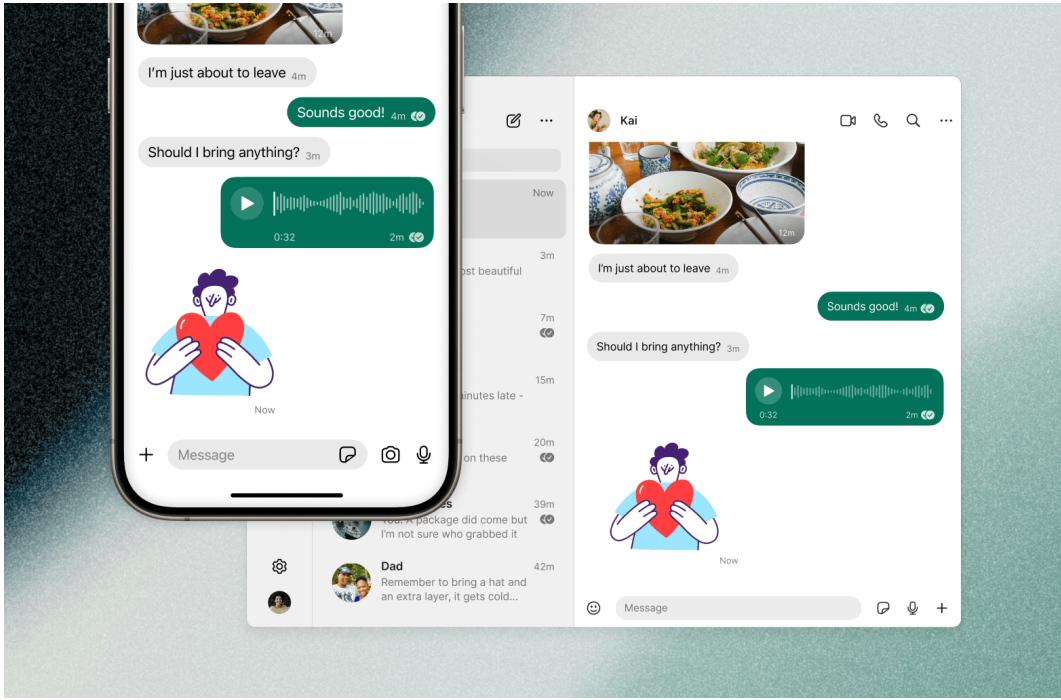
```
0 private async decryptEnvelope(
1     stores: LockedStores,
2     envelope: UnsealedEnvelope,
3     serviceIdKind: ServiceIdKind
4 ): Promise<DecryptResult> {
5   ...
6   const content = Proto.Content.decode(plaintext);
7   if (!wasEncrypted && Bytes.isEmpty(content.decryptionErrorMessage)) {
8     log.warn(
9       `${logId}: dropping plaintext envelope without decryption error message`
10    );
11
12    const { sourceServiceId: senderAci } = envelope;
13    strictAssert(isAciString(senderAci), 'Sender uuid must be an ACI');
14
15    const event = new InvalidPlaintextEvent({
16      senderDevice: envelope.sourceDevice ?? 1,
17      senderAci,
18      timestamp: envelope.timestamp,
19    });
20
21    this.removeFromCache(envelope);
22    const envelopeId = getEnvelopeId(envelope);
23    return { plaintext: undefined, envelope };
24  }
25  ...
26  ...
27  return { plaintext, envelope };
28 }
29
30
```

```
0 private async innerHandleContentMessage(
1     incomingEnvelope: UnsealedEnvelope,
2     plaintext: Uint8Array
3 ): Promise<void> {
4     const content = Proto.Content.decode(plaintext);
5     const envelope = await this.maybeUpdateTimestamp(incomingEnvelope);
6
7     if (
8       content.decryptionErrorMessage &&
9       Bytes.isNotEmpty(content.decryptionErrorMessage)
10    ) { ... }
11    if (content.syncMessage) { ... }
12    if (content.dataMessage) { ... }
13    if (content.nullMessage) { ... }
14    if (content.callingMessage) { ... }
15    if (content.receiptMessage) { ... }
16    if (content.typingMessage) { ... }
17    if (content.storyMessage) { ... }
18    if (content.editMessage) { ... }
19    this.removeFromCache(envelope);
20    if (Bytes.isEmpty(content.senderKeyDistributionMessage)) {
21      throw new Error('Unsupported content message');
22    }
23  }
```

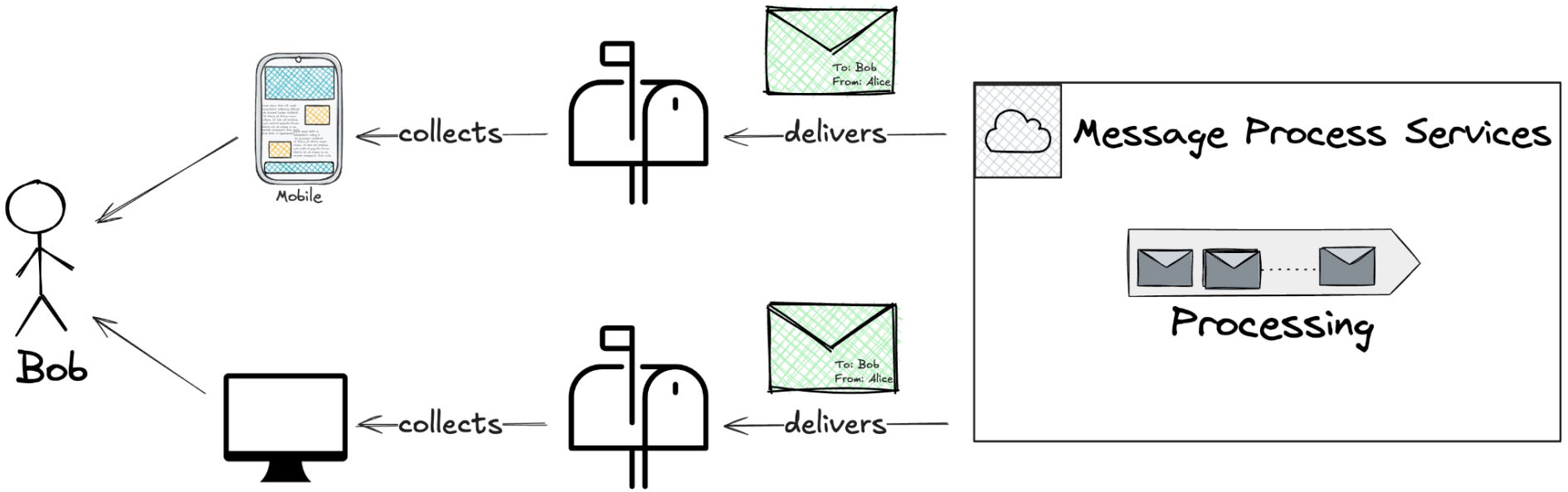
Recap of the vulnerabilities

- Signal allows plain-text messages only for specific error reporting cases
- However, **some clients failed to enforce this, accepting actual text messages** via this fallback path
- This opened the door for a malicious server to inject messages (relevant to SIM swap scenarios too)

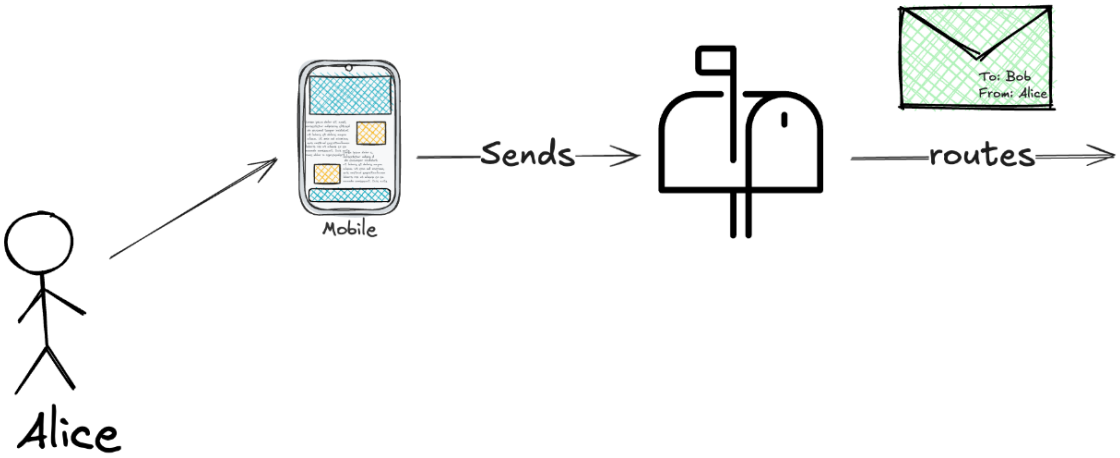
Linked Device

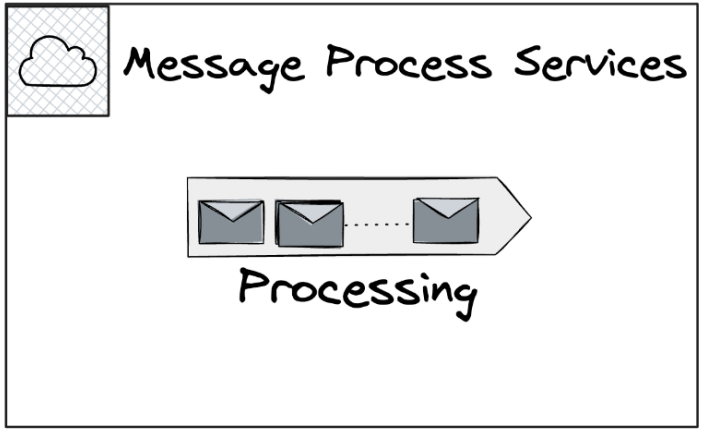


https://signal.org/blog/a-synchronized-start-for-linked-devices/

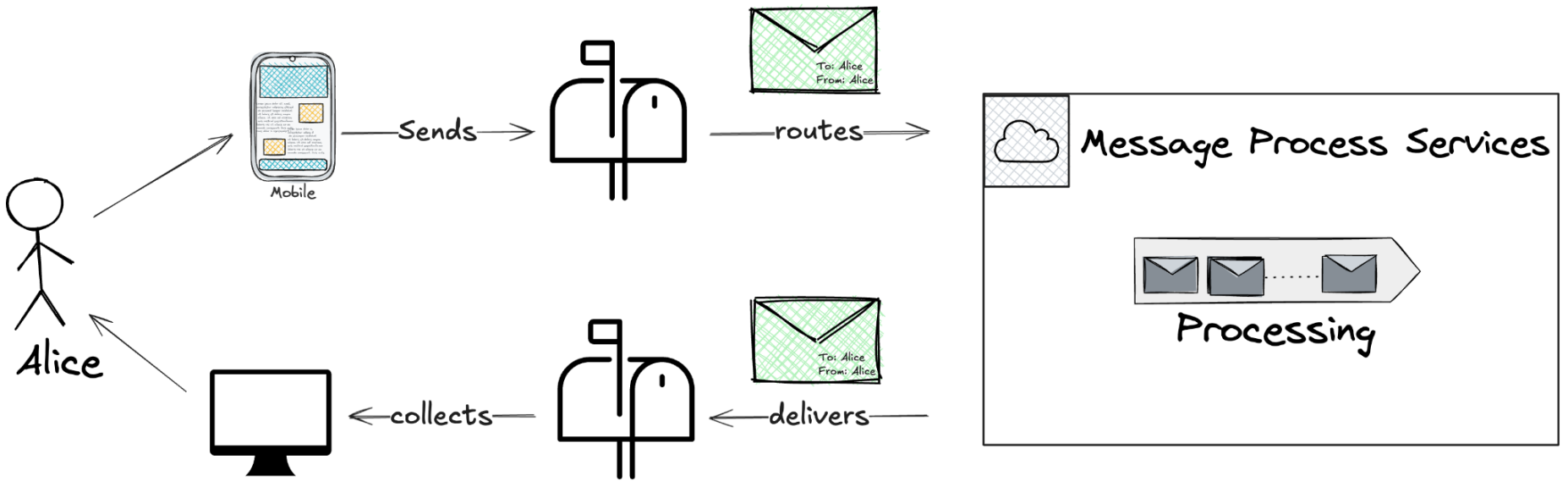


https://signal.org/blog/a-synchronized-start-for-linked-devices/





https://signal.org/blog/a-synchronized-start-for-linked-devices/


Mobile
Sends
To: Alice
From: Alice
routes
Message Process Services
Processing
Alice
collects
To: Alice
From: Alice
delivers

Linked Device - Synchronization

- E2EE between the 2 devices using Signal Protocol
- Same path as 1:1 message
- Protobuf Sync Messages

```
 0 message Content {
 1   optional DataMessage         dataMessage                = 1;
 2   optional SyncMessage         syncMessage                = 2;
 3   optional CallMessage         callMessage                = 3;
 4   optional NullMessage         nullMessage                = 4;
 5   optional ReceiptMessage      receiptMessage             = 5;
 6   optional TypingMessage       typingMessage              = 6;
 7   optional bytes               senderKeyDistributionMessage = 7;
 8   optional bytes               decryptionErrorMessage     = 8;
 9   optional StoryMessage        storyMessage               = 9;
10   optional PniSignatureMessage pniSignatureMessage        = 10;
11   optional EditMessage         editMessage                = 11;
12 }
```

```
 0 message SyncMessage {
 1   optional Sent                    sent                    = 1;
 2   optional Contacts                contacts                = 2;
 3   reserved /* groups */ 3;
 4   optional Request                 request                 = 4;
 5   repeated Read                    read                    = 5;
 6   optional Blocked                 blocked                 = 6;
 7   optional Verified                verified                = 7;
 8   optional Configuration           configuration           = 9;
 9   optional bytes                   padding                 = 8;
10   repeated StickerPackOperation    stickerPackOperation    = 10;
11   optional ViewOnceOpen            viewOnceOpen            = 11;
12   optional FetchLatest             fetchLatest             = 12;
13   optional Keys                    keys                    = 13;
14   optional MessageRequestResponse  messageRequestResponse  = 14;
15   reserved                                                   15;
16   repeated Viewed                  viewed                  = 16;
17   reserved                                                   17;
18   optional PniChangeNumber         pniChangeNumber         = 18;
19   optional CallEvent               callEvent               = 19;
20   optional CallLinkUpdate          callLinkUpdate          = 20;
21   optional CallLogEvent            callLogEvent            = 21;
22   optional DeleteForMe             deleteForMe             = 22;
23 }
```

```
0 message SyncMessage {
1   optional Sent                    sent
2   optional Contacts                contacts
3   reserved /* groups */ 3;
4   optional Request                 request
                                     read
ked                                  blocked
fied                                 verified
iguration                            configuration
s                                    padding
kerPackOperation                     stickerPackOperation
OnceOpen                             viewOnceOpen
hLatest                              fetchLatest
                                     keys
ageRequestResponse                   messageRequestResponse

ed                                   viewed

hangeNumber                          pniChangeNumber
Event                                callEvent
LinkUpdate                           callLinkUpdate
LogEvent                             callLogEvent
teForMe                              deleteForMe
```

```
0 message Sent {
1   optional string destination             = 1;
2   optional string destinationServiceId    = 7;
3   optional uint64 timestamp               = 2;
4   optional DataMessage message            = 3;
5   optional uint64 expirationStartTimestamp = 4;
6   repeated UnidentifiedDeliveryStatus unidentifiedStatus       = 5;
7   optional bool                       isRecipientUpdate        = 6
8   optional StoryMessage storyMessage = 8;
9   repeated StoryMessageRecipient storyMessageRecipients = 9;
10  optional EditMessage editMessage = 10;
11 }
```

```
0 message Read {
1   optional string sender      = 1;
2   optional string senderAci   = 3;
3   optional uint64 timestamp   = 2;
4 }
```

```
0 message SyncMessage {
1   optional Sent                    sent
2   optional Contacts                contacts
3   reserved /* groups */ 3;
4   optional Request                 request
5   repeated Read                    read
6   optional Blocked                 blocked
7   optional Verified                verified
8   optional Configuration           configuration
9   optional bytes                   padding
10  repeated StickerPackOperation    stickerPackOperation
11  optional ViewOnceOpen            viewOnceOpen
12  optional FetchLatest             fetchLatest
13  optional Keys                    keys
14  optional MessageRequestResponse  messageRequestResponse
15  reserved
16  repeated Viewed                  viewed
17  reserved
18  optional PniChangeNumber         pniChangeNumber
19  optional CallEvent               callEvent
20  optional CallLinkUpdate          callLinkUpdate
21  optional CallLogEvent            callLogEvent
22  optional DeleteForMe             deleteForMe
23 }
```

```
0 message Configuration {
1   optional bool   readReceipts                  = 1;
2   optional bool   unidentifiedDeliveryIndicators = 2;
3   optional bool   typingIndicators              = 3;
4   reserved /* linkPreviews */                     4;
5   optional uint32 provisioningVersion           = 5;
6   optional bool   linkPreviews                  = 6;
7 }
```

```
0 message SyncMessage {
1   optional Sent                    sent
2   optional Contacts                contacts
3   reserved /* groups */ 3;
4   optional Request                 request
5   repeated Read                    read
6   optional Blocked                 blocked
7   optional Verified                verified
8   optional Configuration           configuration
9   optional bytes                   padding
10  repeated StickerPackOperation    stickerPackOperation
11  optional ViewOnceOpen            viewOnceOpen
12  optional FetchLatest             fetchLatest
13  optional Keys                    keys
14  optional MessageRequestResponse  messageRequestResponse
15  reserved
16  repeated Viewed                  viewed
17  reserved
18  optional PniChangeNumber         pniChangeNumber
19  optional CallEvent               callEvent
20  optional CallLinkUpdate          callLinkUpdate
21  optional CallLogEvent            callLogEvent
22  optional DeleteForMe             deleteForMe
23 }
```

Processing SyncMessages

- Decrypt using Signal protocol
- Validate sender is self
- Process

```
 0 private func handleIncomingEnvelope(
 1     request: MessageReceiverRequest,
 2     syncMessage: SSKProtoSyncMessage,
 3     localIdentifiers: LocalIdentifiers,
 4     tx: SDSAnyWriteTransaction
 5 ) {
 6     let decryptedEnvelope = request.decryptedEnvelope
 7
 8     guard decryptedEnvelope.sourceAci == localIdentifiers.aci
 9     else {
10         // Sync messages should only come from linked devices.
11         owsFailDebug("Received message from another user.")
12         return
13     }
14     ...
15     ...
```

https://github.com/signalapp/Signal-iOS/blob/4f272777f6e7c1de0f9f025b23e789fac99036e2/SignalServiceKit/Messages/MessageReceiver.swift#L366

```
0 private async handleSyncMessage(
1     envelope: UnsealedEnvelope,
2     syncMessage: ProcessedSyncMessage
3   ): Promise<void> {
4     const ourNumber = this.storage.user.getNumber();
5     const ourAci = this.storage.user.getCheckedAci();
6
7     const fromSelfSource = envelope.source && envelope.source === ourNumber;
8     const fromSelfSourceUuid =
9       envelope.sourceServiceId && envelope.sourceServiceId === ourAci;
10    if (!fromSelfSource && !fromSelfSourceUuid) {
11      throw new Error('Received sync message from another number');
12    }
13    ...
14    ...
```

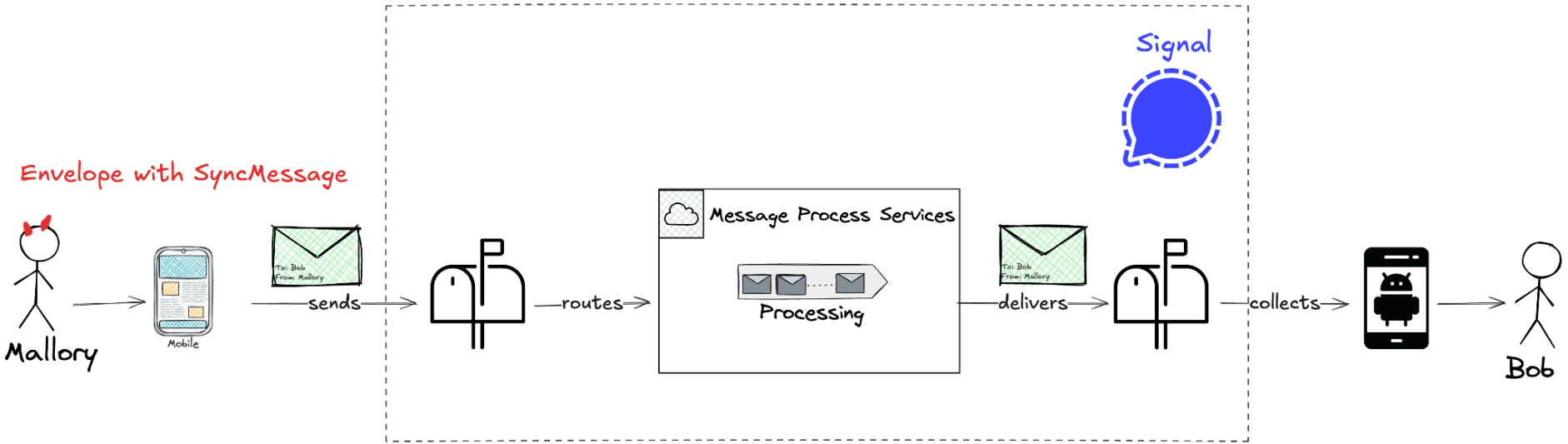
https://github.com/signalapp/Signal-Desktop/blob/2a55bfbef93f75f8b40f32a002e06827b15ffe91/ts/textsecure/MessageReceiver.ts#L3041

```kotlin
private fun validateSyncMessage(envelope: Envelope, syncMessage: SyncMessage): Result {
  if (syncMessage.sent != null)
    { ... }

  if (syncMessage.read.any { it.senderAci.isNullOrInvalidAci() })
    { ... }

  if (syncMessage.viewed.any { it.senderAci.isNullOrInvalidAci() })
    { ... }

  if (syncMessage.viewOnceOpen != null && syncMessage.viewOnceOpen.senderAci.isNullOrInvalidAci())
    { ... }

  if (syncMessage.verified != null && syncMessage.verified.destinationAci.isNullOrInvalidAci())
    { ... }

  if (syncMessage.stickerPackOperation.any { it.packId == null })
    { ... }

  if (syncMessage.blocked != null && syncMessage.blocked.acis.any { it.isNullOrInvalidAci() })
    { ... }
  ...
  ...

  return Result.Valid
}
```

https://github.com/signalapp/Signal-Android/blob/d84612ebf41a2f9c45f54217219db21960605c98/libsignal-service/src/main/java/org/whispersystems/signalservice/api/messages/EnvelopeContentValidator.kt#L148

Forging SyncMessages

- Missing sender validation on SyncMessages
- Android-only vulnerability
- Impact
  - Attackers can send Sync messages to Android clients (0-click)
  - Actions possible: send, delete, mark as read, update settings
- Reading message content was **not** possible
- Patched immediately, fixed in September 2024


Envelope with SyncMessage
Signal
Mallory
Mobile
To: Bob
From: Mallory
sends
routes
Message Process Services
Processing
To: Bob
From: Mallory
delivers
collects
Bob

Demo

```
 0 message SyncMessage {
 1   optional Sent                    sent                    = 1;
 2   optional Contacts                contacts                = 2;
 3   reserved /* groups */ 3;
 4   optional Request                 request                 = 4;
 5   repeated Read                    read                    = 5;
 6   optional Blocked                 blocked                 = 6;
 7   optional Verified                verified                = 7;
 8   optional Configuration           configuration           = 9;
 9   optional bytes                   padding                 = 8;
10   repeated StickerPackOperation    stickerPackOperation    = 10;
11   optional ViewOnceOpen            viewOnceOpen            = 11;
12   optional FetchLatest             fetchLatest             = 12;
13   optional Keys                    keys                    = 13;
14   optional MessageRequestResponse  messageRequestResponse  = 14;
15   reserved                                                   15;
16   repeated Viewed                  viewed                  = 16;
17   reserved                                                   17;
18   optional PniChangeNumber         pniChangeNumber         = 18;
19   optional CallEvent               callEvent               = 19;
20   optional CallLinkUpdate          callLinkUpdate          = 20;
21   optional CallLogEvent            callLogEvent            = 21;
22   optional DeleteForMe             deleteForMe             = 22;
23 }
```

```
0 private fun validateSyncMessage(envelope: Envelope, syncMessage: SyncMessage, localAci: ACI):
  Result {
1   // Source serviceId was already determined to be a valid serviceId in general
2   val sourceServiceId = ServiceId.parseOrThrow(envelope.sourceServiceId!!)
3
4   if (sourceServiceId != localAci) {
5     return Result.Invalid("[SyncMessage] Source was not our own account!")
6   }
7
8   if (syncMessage.sent != null) { ... }
9
10  if (syncMessage.read.any { it.senderAci.isNullOrInvalidAci() }) { ... }
11  ...
12  ...
```

Fix commit: https://github.com/signalapp/Signal-Android/commit/ac10ff4cbe1312ce4932337278d3c4a2c69954cc

Vulnerability Recap

- When you have a linked devices, your devices synchronize the state via an invisible E2EE “conversation” between them
- This conversation use special message type “SyncMessages”
- The vulnerability was that Android clients accepted SyncMessages not just from linked devices but **from anyone** (even if you had no linked devices)

Forging SyncMessages

- CVE-2025-24903: Whisperfish an unofficial signal clients on top of Signal library
- CVSS score: 8.5

What Else Did I Review?

- Language specific
- Application-specific
- Logic-based
- Product-specific

Wrapping up

- A security engineer review on Signal
- Close collaboration with the Signal team
- Takeaways:
  - How signal works?
  - Privacy guarantees
  - Vulnz (all fixed)

Big thanks to the team

- Jim O'Leary (Signal VP of engineering)
- Ehren Kret (Signal CTO)
- Signal engineering and comms teams
- Otto Ebeling
- Edoardo Nava