WatchWitch

Hacking the Apple Watch

TECHNISCHE UNIVERSITÄT DARMSTADT

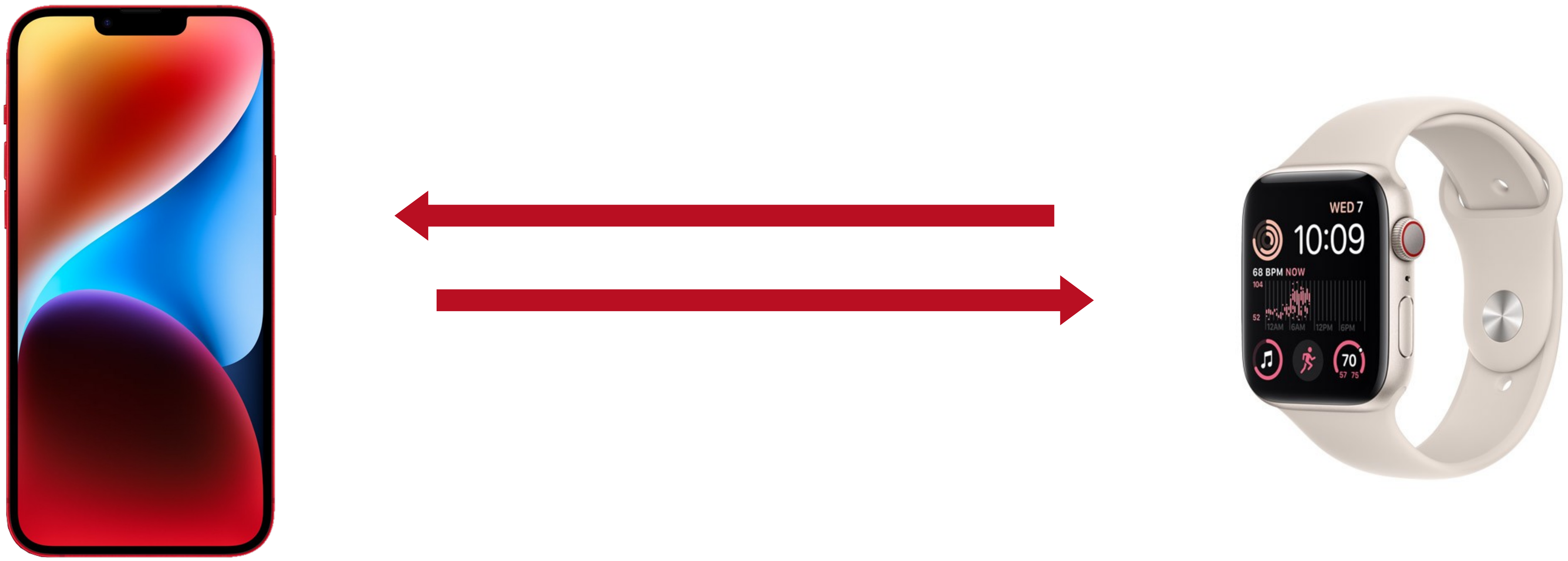
1. Prelude

What?

What?


GPS
ECG
Noise Levels
Heart Rate
Blood Oxygen Saturation
Motion Tracking
Skin Temperature
WED 7
10:09
68 BPM NOW

What?



What?

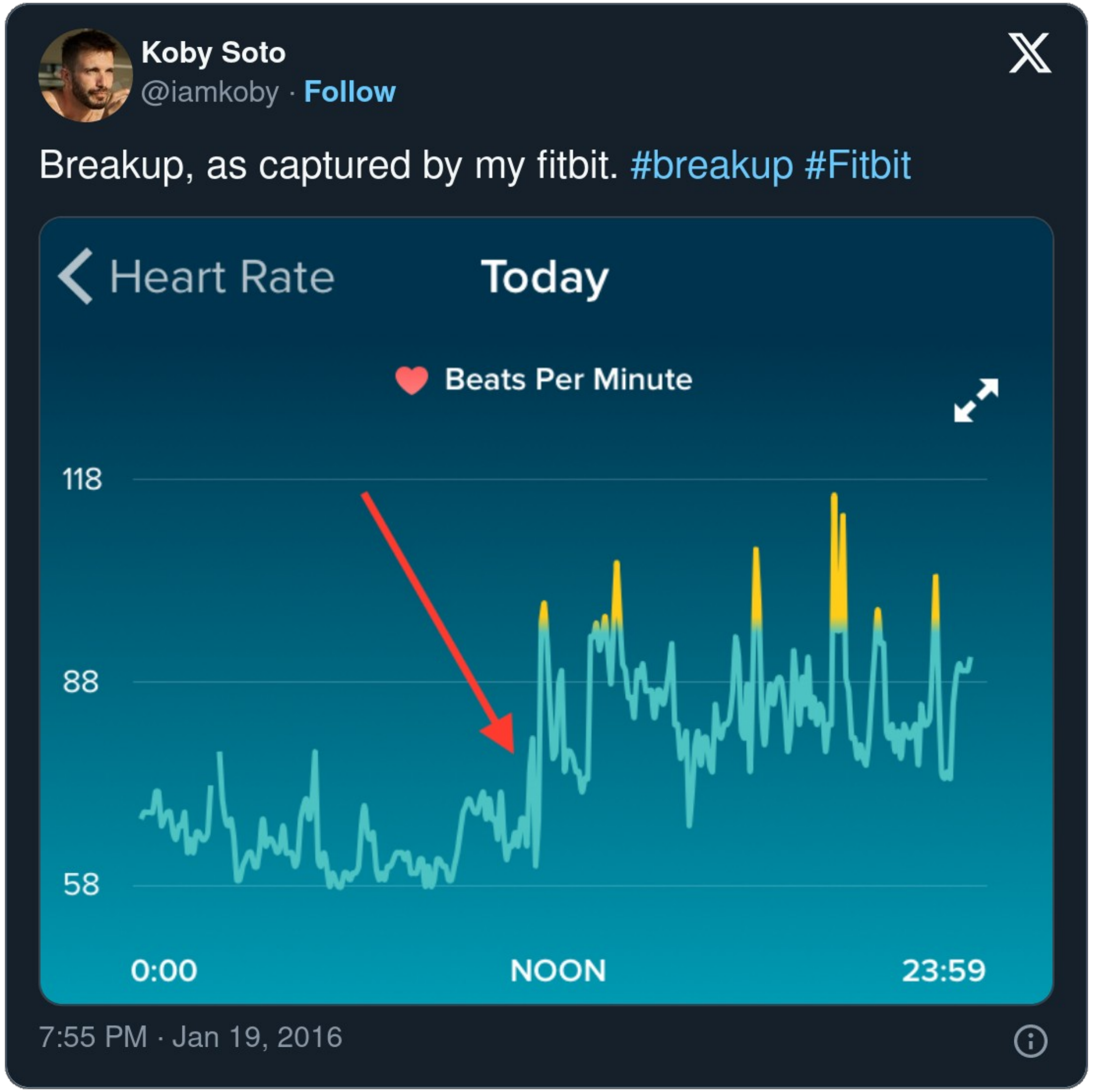
iMessage
SMS
calls
3rd party notifications

heart rate
sleep tracking
cycle tracking
workouts
location
ECG

mail
passbook
gallery
contacts
reminders
calendar

settings
...
...
...

Why should I care?



twitter.com/iamkoby/status/689521611611971588


Browse
Search
Health Categories
Activity
Body Measurements
Cycle Tracking
Hearing
Respiratory
Sleep
Symptoms
Vitals
Other Data
Summary
Browse

```
INSERT INTO quantity_samples VALUES(48802,0.42,NULL,NULL);
INSERT INTO quantity_samples VALUES(48803,0.39,NULL,NULL);
INSERT INTO quantity_samples VALUES(48804,0.98,59.0,1);
INSERT INTO quantity_samples VALUES(48824,0.65,NULL,NULL);
INSERT INTO quantity_samples VALUES(48825,12.2,NULL,NULL);
INSERT INTO quantity_samples VALUES(48858,1.35,81.0,1)

sqlite> SELECT private_classification, average_heart_rate,
hex(voltage_payload) FROM ecg_samples LIMIT 1;
private_classification|average_heart_rate|voltage_payload
3|76.0|0A8AD70408011D6CFC6DC21DE17991C21D02EFA9C21D5A52C0C
21DD41BDAC21D4653F2C21D1F93FAC21D94EBE6C21DD5F6A9C21DA9520
BC21D4C30CD411D07A2AF421D649A0E431D3A2239431D4A1658431D1A6
56B431D068473431D308970431DB21261431D7EE045431DBA6721431D5
0EFED421DDB5096421D632E07421DD8A539C01DC5F204C21D28B15F...
```

2. Within the Watch

ok cool

we started working out

NanoSyncMessage
version: 12

NanoSyncChangeSet
date: Tue Dec 19 03:00:33 CET 2023,
status: Continue

NanoSyncChange
type: Quantity Sample
sync anchor: 11 start: 402 end: 412

ObjectCollection

- **Source:** Apple Watch / com.apple.health
- **Provenance:** Watch5,1 v7.3.3 18S830, Europe/Berlin
- **QuantitySample:** BasalEnergyBurned 63.69 kcal
- ...

we started working out

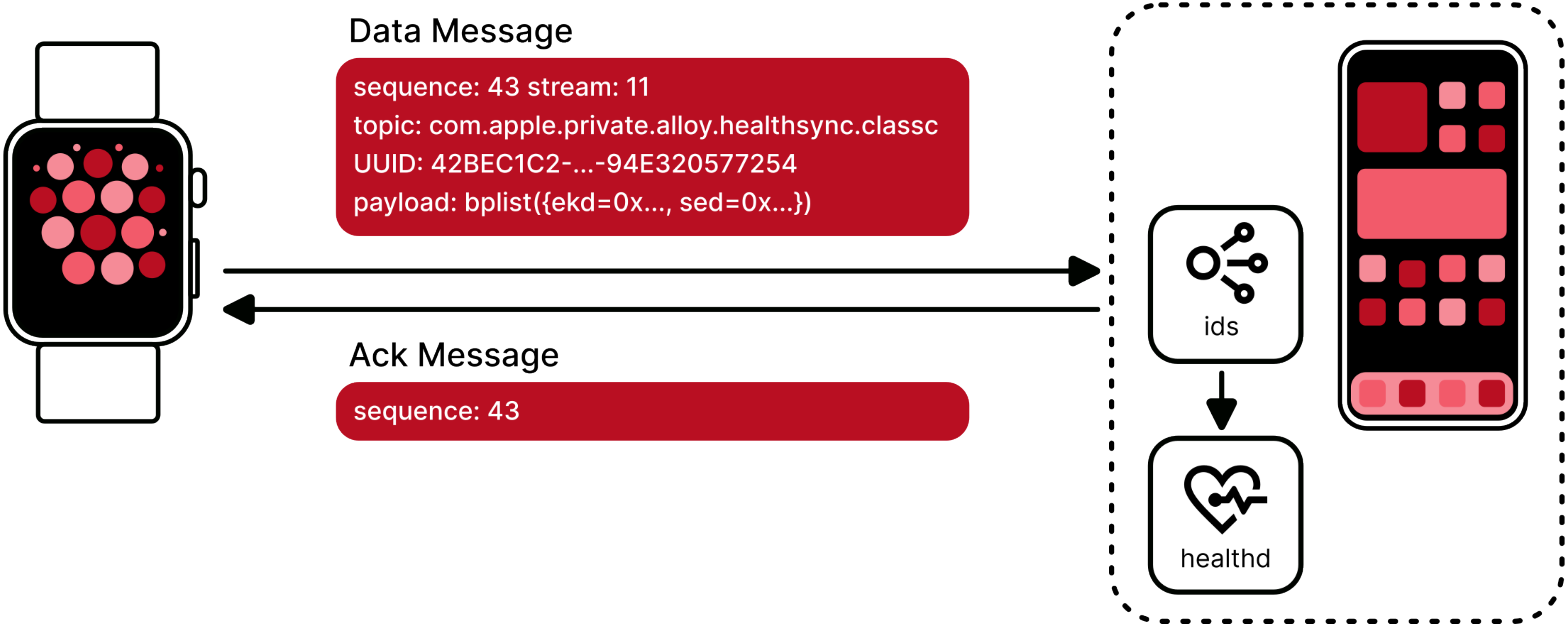
NanoSyncMessage
version: 12

SyncStatus
status: Continue

- **Sync Anchor:** anchor 11 value 412
- **Sync Anchor:** anchor 16 value 103
- **Sync Anchor:** anchor 21 value 67
- ...

ok cool

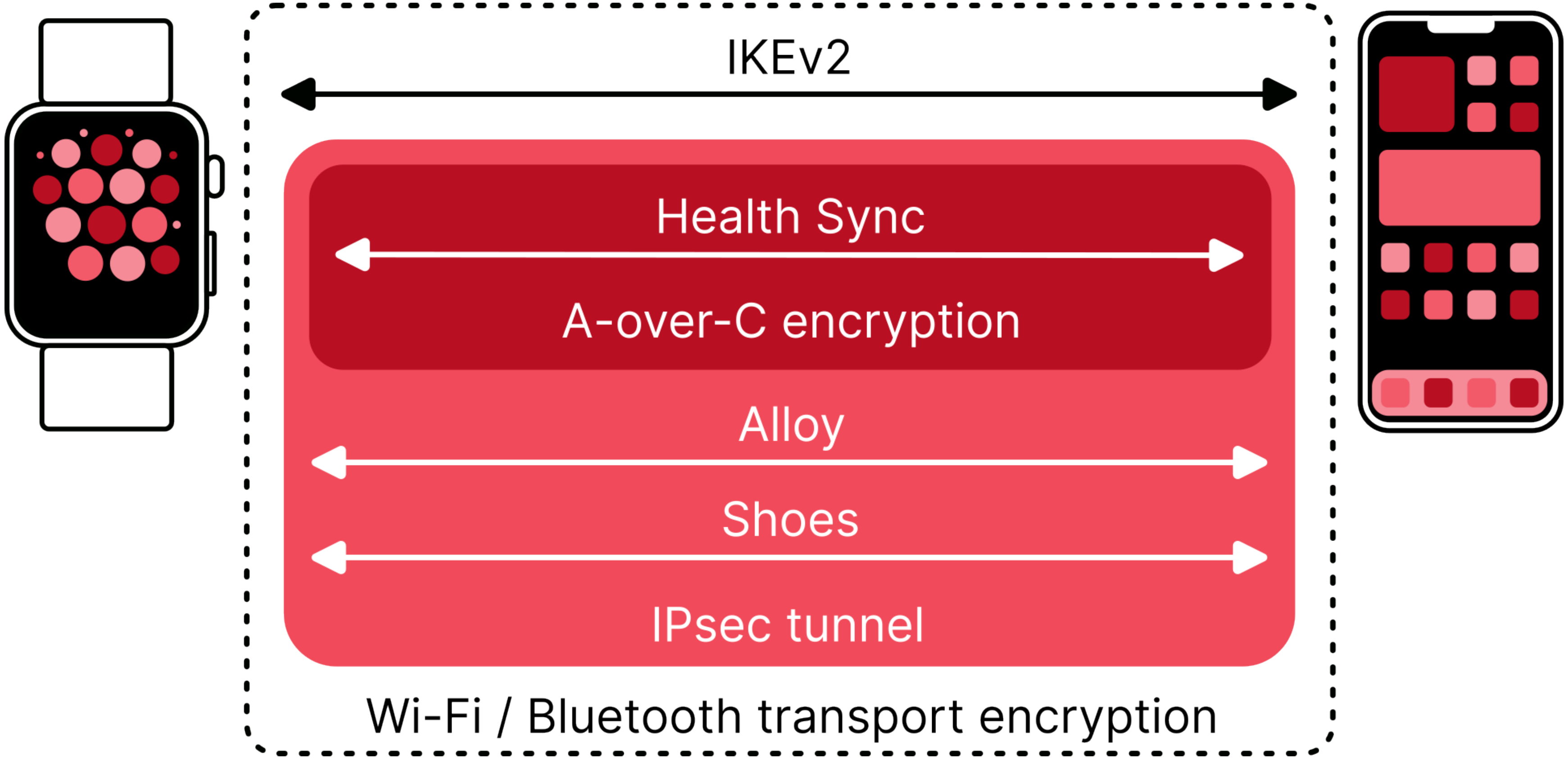
Alloy


Data Message
sequence: 43 stream: 11
topic: com.apple.private.alloy.healthsync.classc
UUID: 42BEC1C2-...-94E320577254
payload: bplist({ekd=0x..., sed=0x...})
Ack Message
sequence: 43
ids
healthd

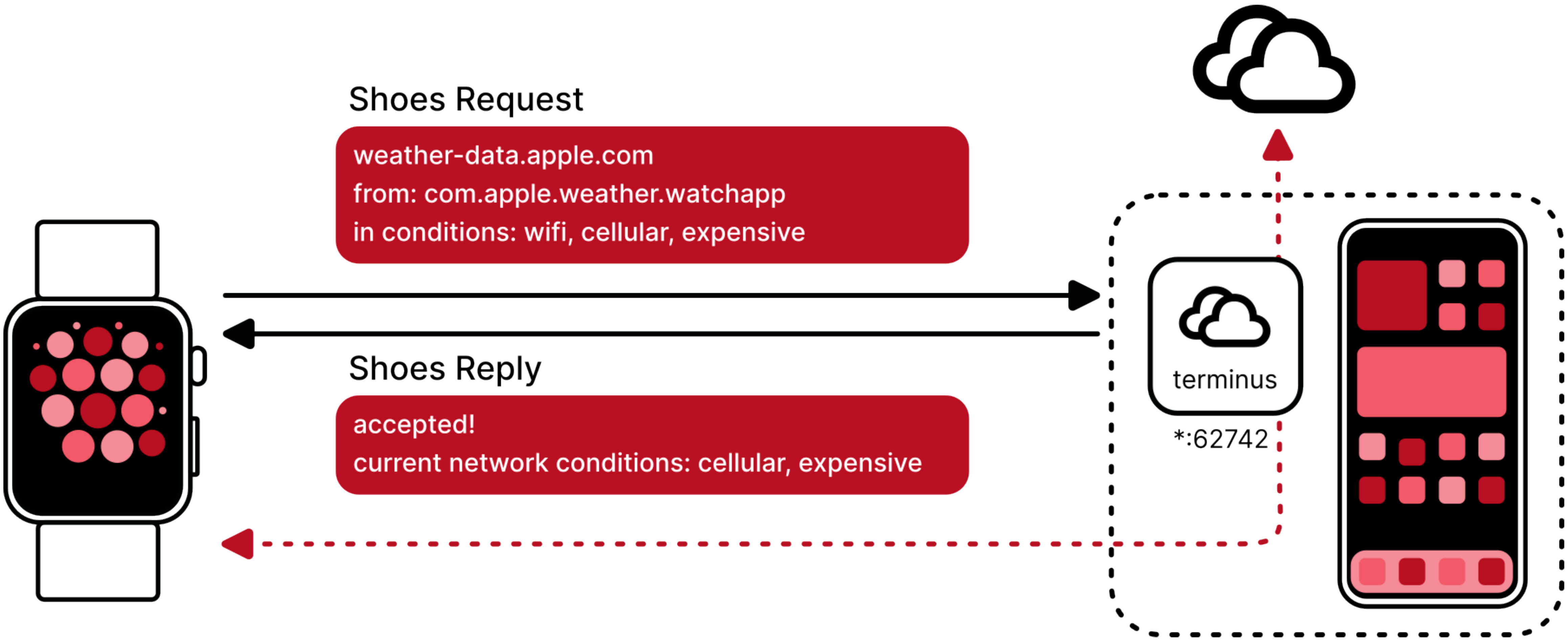
Alloy

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
type	length				sequence				stream		flags	len (resp. id.)			
ASCII response identifier ...															
len (msg UUID)				ASCII message UUID ...											
len (topic)				ASCII topic ...											
... payload ...															
expiry date															

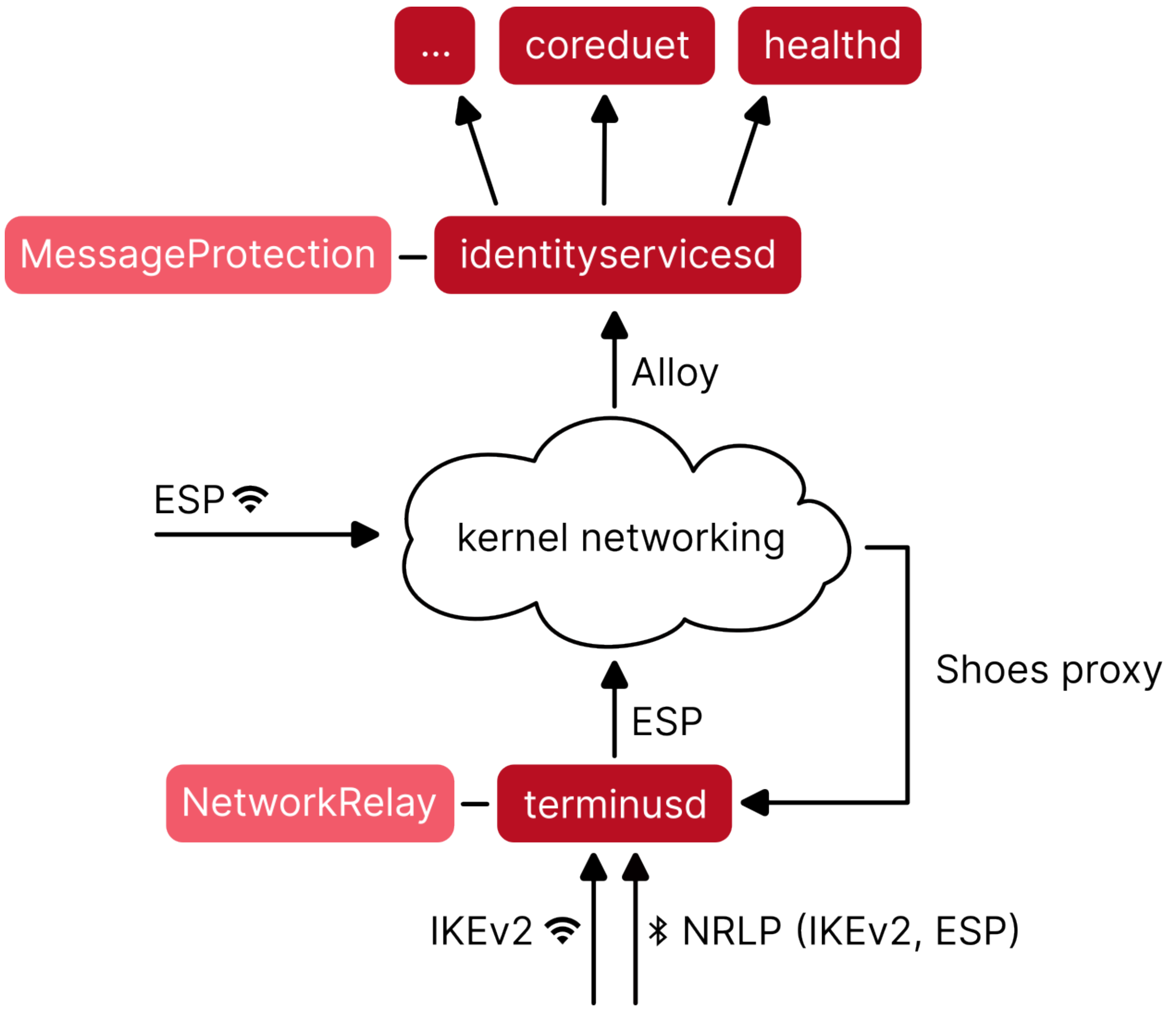
Tunnels


IKEv2
Health Sync
A-over-C encryption
Alloy
Shoes
IPsec tunnel
Wi-Fi / Bluetooth transport encryption

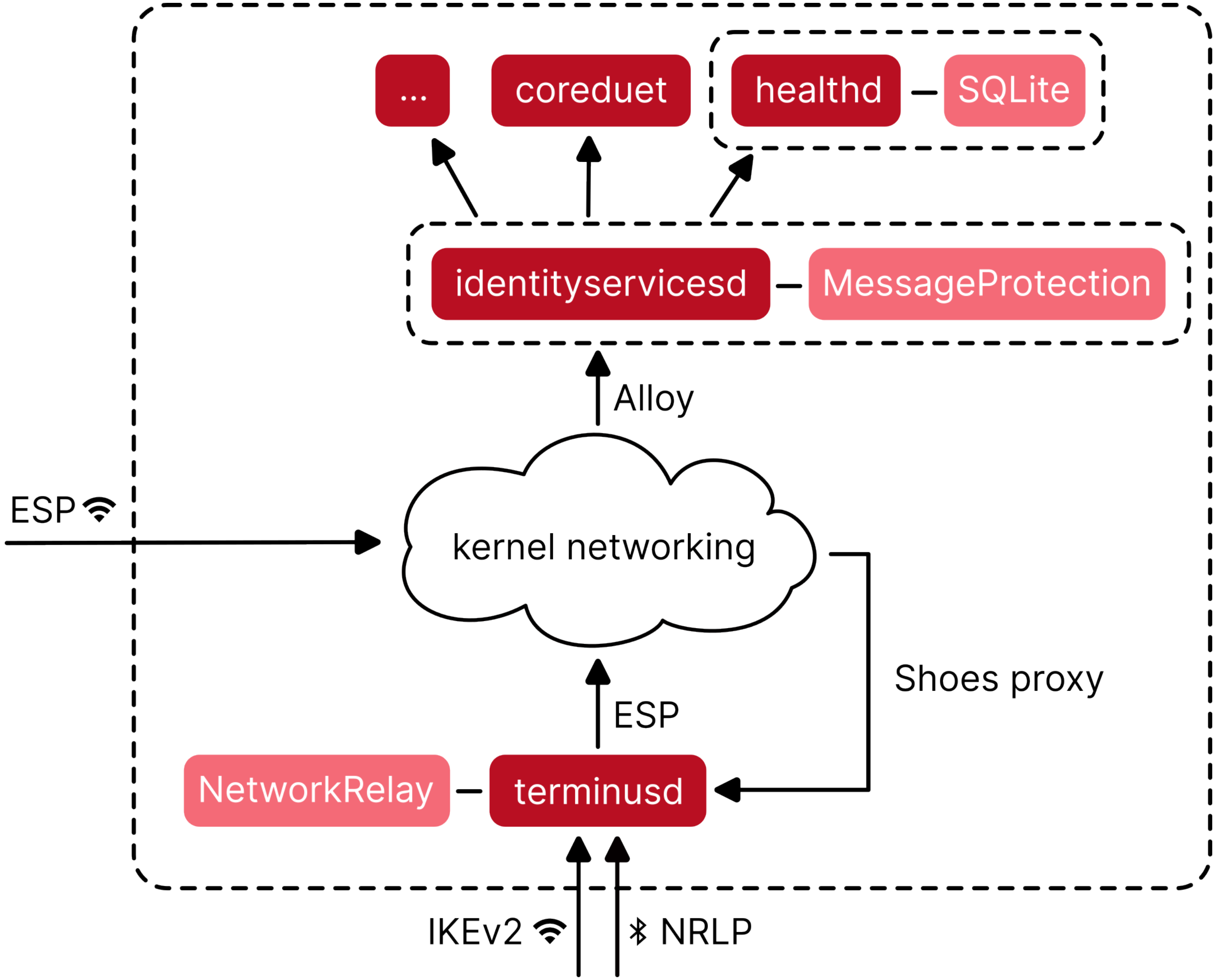
Shoes


Shoes Request
weather-data.apple.com
from: com.apple.weather.watchapp
in conditions: wifi, cellular, expensive
Shoes Reply
accepted!
current network conditions: cellular, expensive
terminus
*:62742

Message Flow

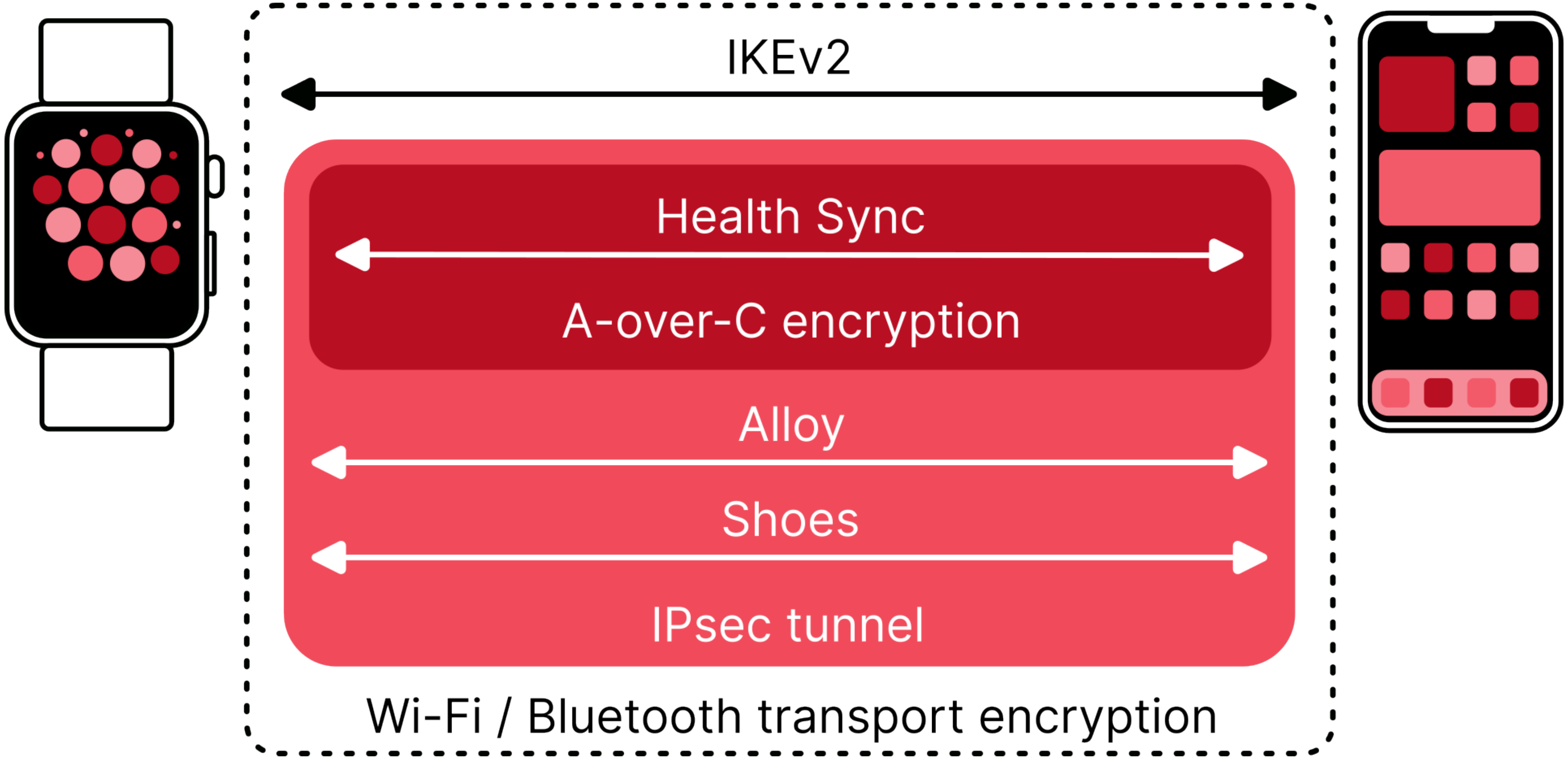


3. Attack!



trust boundaries in message handling

Re: Tunnels


IKEv2
Health Sync
A-over-C encryption
Alloy
Shoes
IPsec tunnel
Wi-Fi / Bluetooth transport encryption

IKEv2 Handling

> **3.10. Notify Payload**
> The Notify payload, denoted N in this document, is used to transmit informational data, such as error conditions and state transitions, to an IKE peer. A Notify payload may appear in a response message (usually specifying why a request was rejected), in an INFORMATIONAL exchange (to report an error not in an IKE request), or in any other message to indicate sender capabilities or to modify the meaning of the request.
>
> — *RFC 7296*

48603	Device name	e.g. "iPhone", "Apple Watch"
48604	Build version	e.g. "18H17", "18S830"
50701	ProxyNotify	IPv6 address and port of SHOES server on the phone
50702	LinkDirectorMessage	used for link state signaling and WiFi discovery

Apple-defined private notify types

Type	Name	Comment
1	Hello	no payload, signals restart
2	UpdateWiFiAddressIPv6	2 byte port followed by 16 byte IP
3	UpdateWiFiAddressIPv4	2 byte port followed by 4 byte IP
4	UpdateWiFiSignature	variable length, unused?
5	PreferWiFi	no payload
6	DeviceLinkState	1 byte preferred link, 1: Bluetooth, 2: WiFi

IKEv2 Handling

> 3.10. **Notify Payload**
> The Notify payload, denoted N in this document, is used to transmit informational data, such as error conditions and state transitions, to an IKE peer. A Notify payload may appear in a response message (usually specifying why a request was rejected), in an INFORMATIONAL exchange (to report an error not in an IKE request), or in any other message to indicate sender capabilities or to modify the meaning of the request.
>
> — *RFC 7296*

48603	Device name	e.g. "iPhone", "Apple Watch"
48604	Build version	e.g. "18H17", "18S830"
50701	ProxyNotify	IPv6 address and port of SHOES server on the phone
50702	LinkDirectorMessage	used for link state signaling and WiFi discovery

Apple-defined private notify types

Type	Name	Comment
1	Hello	no payload, signals restart
2	UpdateWiFiAddressIPv6	2 byte port followed by 16 byte IP
3	UpdateWiFiAddressIPv4	2 byte port followed by 4 byte IP
4	UpdateWiFiSignature	variable length, unused?
5	PreferWiFi	no payload
6	DeviceLinkState	1 byte preferred link, 1: Bluetooth, 2: WiFi

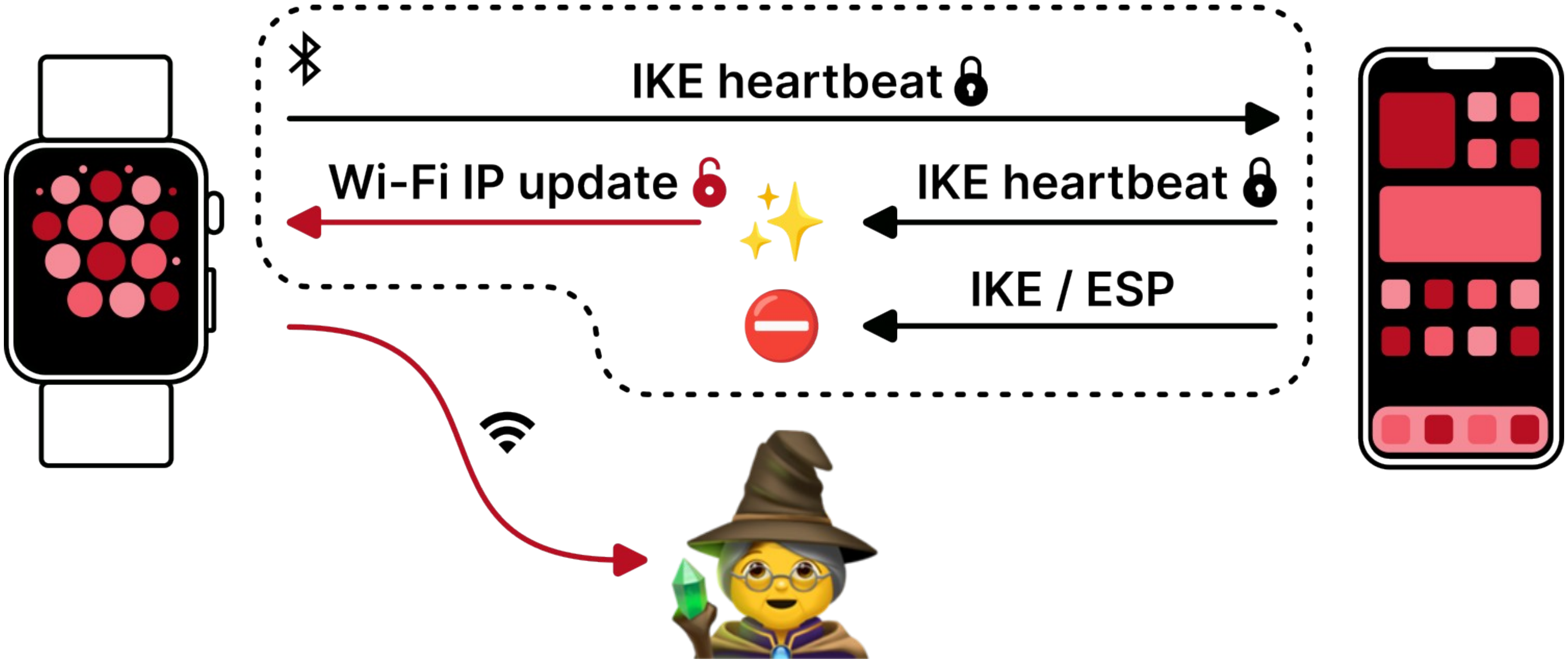
IKEv2 Handling

> 3.10. **Notify Payload**
>
> The Notify payload, denoted N in this document, is used to transmit informational data, such as error conditions and state transitions, to an IKE peer. A Notify payload may appear in a response message (usually specifying why a request was rejected), in an INFORMATIONAL exchange (to report an error not in an IKE request), or in any other message to indicate sender capabilities or to modify the meaning of the request.
>
> — *RFC 7296*

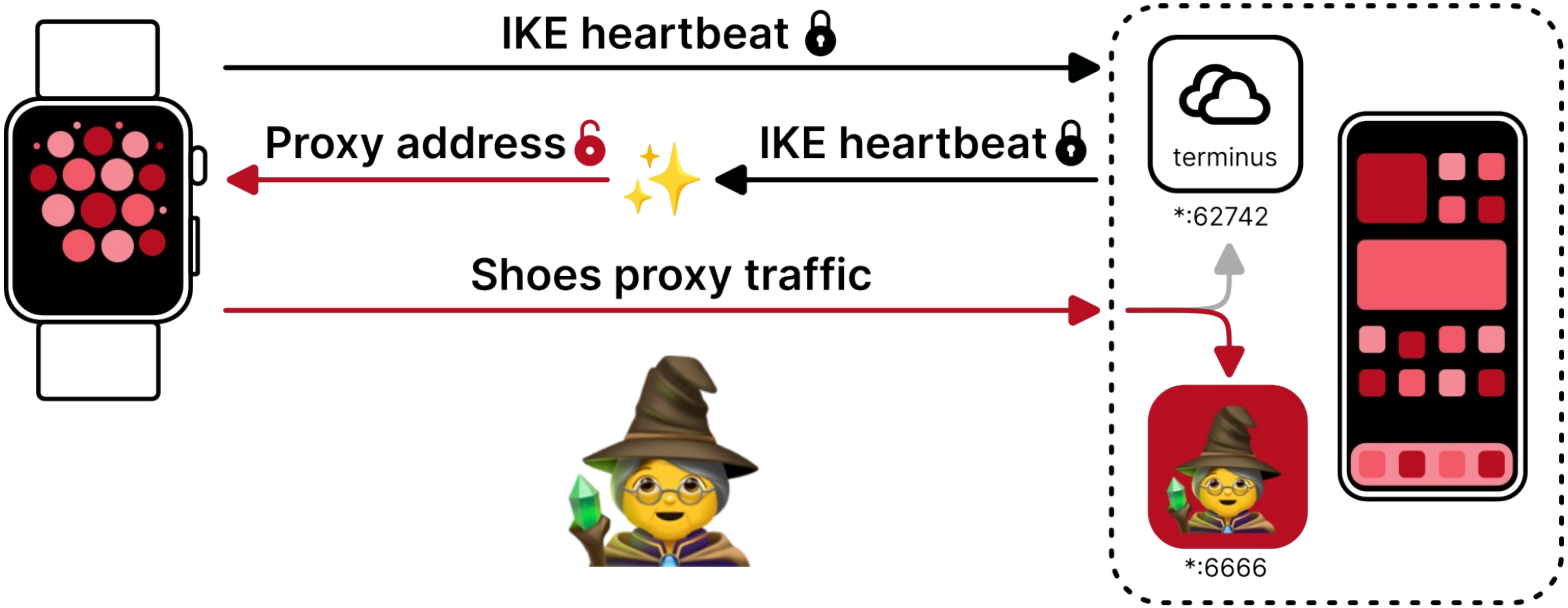
48603	Device name	e.g. "iPhone", "Apple Watch"
48604	Build version	e.g. "18H17", "18S830"
50701	ProxyNotify	IPv6 address and port of SHOES server on the phone
50702	LinkDirectorMessage	used for link state signaling and WiFi discovery

Apple-defined private notify types

Type	Name	Comment
1	Hello	no payload, signals restart
2	UpdateWiFiAddressIPv6	2 byte port followed by 16 byte IP
3	UpdateWiFiAddressIPv4	2 byte port followed by 4 byte IP
4	UpdateWiFiSignature	variable length, unused?
5	PreferWiFi	no payload
6	DeviceLinkState	1 byte preferred link, 1: Bluetooth, 2: WiFi

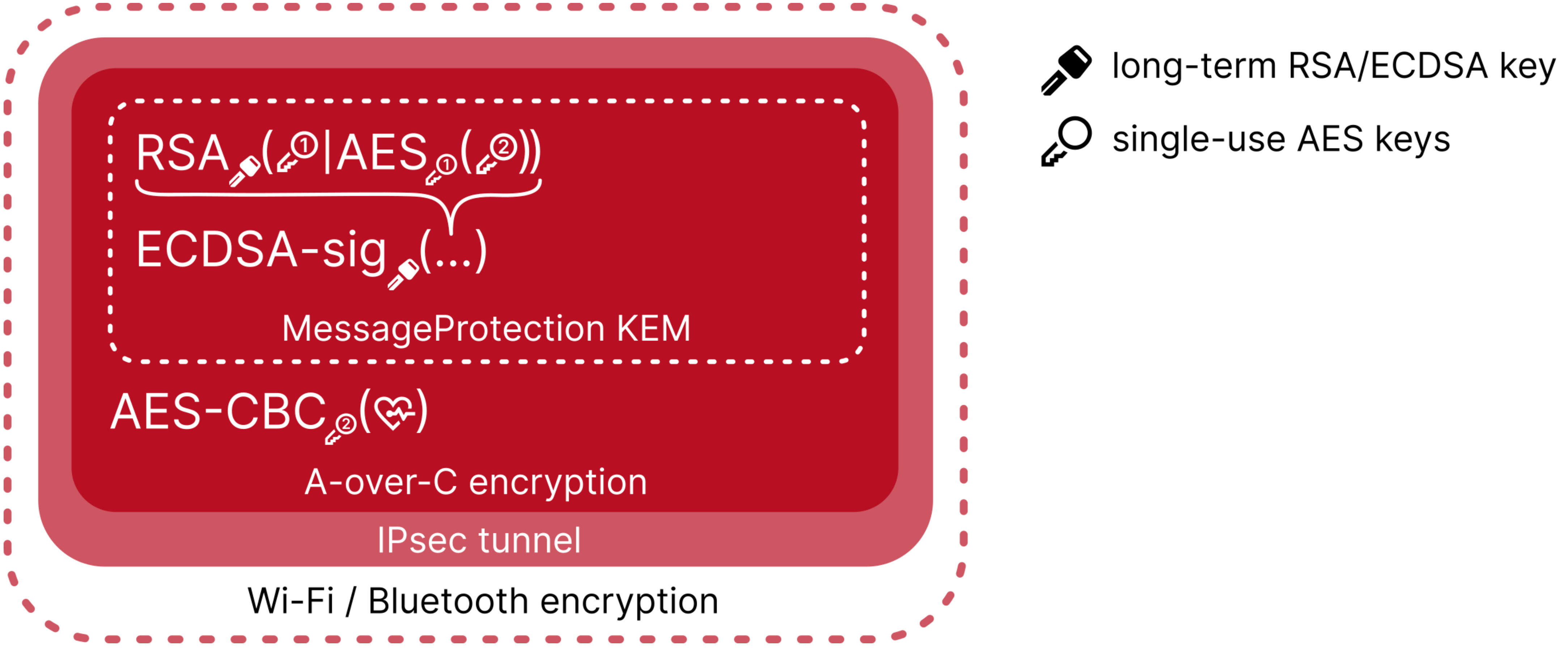


injection of a forged Wi-Fi IP address update



injection of a forged Shoes proxy endpoint

Malleable Encryption



Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage

Christina Garman
Johns Hopkins University
cgarman@cs.jhu.edu

Matthew Green
Johns Hopkins University
mgreen@cs.jhu.edu

Gabriel Kaptchuk
Johns Hopkins University
gkaptchuk@cs.jhu.edu

Ian Miers
Johns Hopkins University
imiers@cs.jhu.edu

Michael Rushanan
Johns Hopkins University
micharu1@cs.jhu.edu

Abstract

Apple's iMessage is one of the most widely-deployed end-to-end encrypted messaging protocols. Despite its broad deployment, the encryption protocols used by iMessage have never been subjected to rigorous cryptanalysis. In this paper, we conduct a thorough analysis of iMessage to determine the security of the protocol against a variety of attacks. Our analysis shows that iMessage has significant vulnerabilities that can be exploited by a sophisticated attacker. In particular, we outline a novel chosen ciphertext attack on Huffman compressed data, which allows *retrospective* decryption of some iMessage payloads in less than 2^{18} queries. The practical implication of these attacks is that any party who gains access to iMessage ciphertexts may potentially decrypt them remotely and after the fact. We additionally describe mitigations that will prevent these attacks on the protocol, without breaking backwards compatibility. Apple has deployed our mitigations in the latest iOS and OS X releases.

1 Introduction

The past several years have seen widespread adoption of end-to-end encrypted text messaging protocols. In this work we focus on one of the most popular such protocols: Apple's iMessage. Introduced in 2011, iMessage is an end-to-end encrypted text messaging system that supports both iOS and OS X devices. While Apple does not provide up-to-date statistics on iMessage usage, in February 2016 an Apple executive noted that the system had a peak transmission rate of more then 200,000 messages per second, across 1 billion deployed devices [12].

The broad adoption of iMessage has been controversial, particularly within the law enforcement and national security communities. In 2013, the U.S. Drug Enforcement Agency deemed iMessage "a challenge for DEA intercept" [22], while in 2015 the U.S. Department of Justice accused Apple of thwarting an investigation by refusing to turn over iMessage plaintext [11]. iMessage has been at the center of a months-long debate initiated by U.S. and overseas officials over the implementation of "exceptional access" mechanisms in end-to-end encrypted communication systems [7, 26, 33], and some national ISPs have temporarily blocked the protocol [32]. Throughout this controversy, Apple has consistently maintained that iMessage encryption is end-to-end and that even Apple cannot recover the plaintext for messages transmitted through its servers [10].

Given iMessage's large installed base and the high stakes riding on its confidentiality, one might expect iMessage to have received critical attention from the research community. Surprisingly, there has been very little analysis of the system, in large part due to the fact that Apple has declined to publish the details of iMessage's encryption protocol. In this paper we aim to remedy this situation. Specifically, we attempt to answer the following question: how secure is Apple iMessage?

Our contributions. In this work we analyze the iMessage protocol and identify several weaknesses that an attacker may use to decrypt iMessages and attachments. While these flaws do not render iMessage completely insecure, some flaws reduce the level of security to that of the TLS encryption used to secure communications between end-user devices and Apple's servers. This finding is surprising given the protection claims advertised by Apple [10]. Moreover, we determine that the flaws we detect in iMessage may have implications for other aspects of Apple's ecosystem, as we discuss below.

To perform our analysis, we derived a specification for iMessage by conducting a partial black-box reverse engineering of the protocol as implemented on multiple iOS and OS X devices. Our efforts extend a high-level protocol overview published by Apple [9] and two existing partial reverse-engineering efforts [1, 34]. Armed with a protocol specification, we conducted manual cryptanal-

Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage

Christina Garman
Johns Hopkins University
cgarman@cs.jhu.edu

Matthew Green
Johns Hopkins University
mgreen@cs.jhu.edu

Gabriel Kaptchuk
Johns Hopkins University
gkaptchuk@cs.jhu.edu

Ian Miers
Johns Hopkins University
imiers@cs.jhu.edu

Michael Rushanan
Johns Hopkins University
micharu1@cs.jhu.edu

Abstract

Apple's iMessage is one of the most widely-deployed end-to-end encrypted messaging protocols. Despite its broad deployment, the encryption protocols used by iMessage have never been subjected to rigorous cryptanalysis. In this paper, we conduct a thorough analysis of iMessage to determine the security of the protocol against a variety of attacks. Our analysis shows that iMessage has significant vulnerabilities that can be exploited by a sophisticated attacker. In particular, we outline a novel chosen ciphertext attack on Huffman compressed data, which allows *retrospective* decryption of some iMessage payloads in less than 2^{18} queries. The practical implication of these attacks is that any party who gains access to iMessage ciphertexts may potentially decrypt them remotely and after the fact. We additionally describe mitigations that will prevent these attacks on the protocol, without breaking backwards compatibility. Apple has deployed our mitigations in the latest iOS and OS X releases.

1 Introduction

The past several years have seen widespread adoption of end-to-end encrypted text messaging protocols. In this work we focus on one of the most popular such protocols: Apple's iMessage. Introduced in 2011, iMessage is an end-to-end encrypted text messaging system that supports both iOS and OS X devices. While Apple does not provide up-to-date statistics on iMessage usage, in February 2016 an Apple executive noted that the system had a peak transmission rate of more then 200,000 messages per second, across 1 billion deployed devices [12].

The broad adoption of iMessage has been controversial, particularly within the law enforcement and national security communities. In 2013, the U.S. Drug Enforcement Agency deemed iMessage "a challenge for DEA intercept" [22], while in 2015 the U.S. Department of Justice accused Apple of thwarting an investigation by refusing to turn over iMessage plaintext [11]. iMessage has been at the center of a months-long debate initiated by U.S. and overseas officials over the implementation of "exceptional access" mechanisms in end-to-end encrypted communication systems [7, 26, 33], and some national ISPs have temporarily blocked the protocol [32]. Throughout this controversy, Apple has consistently maintained that iMessage encryption is end-to-end and that even Apple cannot recover the plaintext for messages transmitted through its servers [10].

Given iMessage's large installed base and the high stakes riding on its confidentiality, one might expect iMessage to have received critical attention from the research community. Surprisingly, there has been very little analysis of the system, in large part due to the fact that Apple has declined to publish the details of iMessage's encryption protocol. In this paper we aim to remedy this situation. Specifically, we attempt to answer the following question: how secure is Apple iMessage?

Our contributions. In this work we analyze the iMessage protocol and identify several weaknesses that an attacker may use to decrypt iMessages and attachments. While these flaws do not render iMessage completely insecure, some flaws reduce the level of security to that of the TLS encryption used to secure communications between end-user devices and Apple's servers. This finding is surprising given the protection claims advertised by Apple [10]. Moreover, we determine that the flaws we detect in iMessage may have implications for other aspects of Apple's ecosystem, as we discuss below.

To perform our analysis, we derived a specification for iMessage by conducting a partial black-box reverse engineering of the protocol as implemented on multiple iOS and OS X devices. Our efforts extend a high-level protocol overview published by Apple [9] and two existing partial reverse-engineering efforts [1, 34]. Armed with a protocol specification, we conducted manual cryptanal-

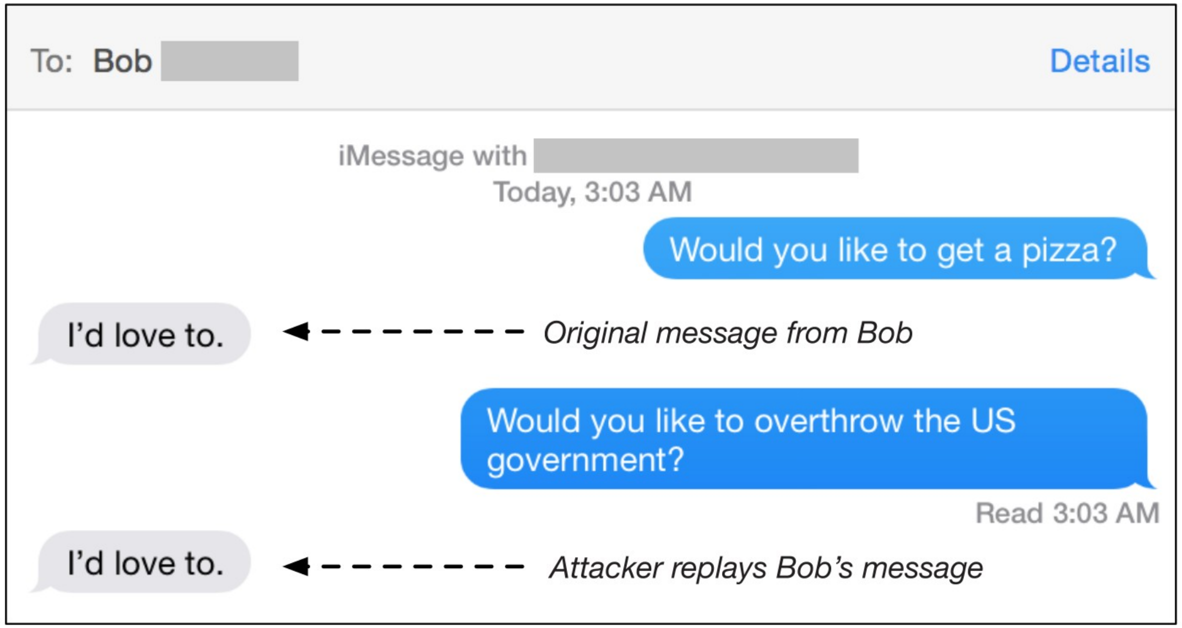


Figure 2: Example of a simple ciphertext replay.

Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage

Christina Garman
Johns Hopkins University
cgarman@cs.jhu.edu

Matthew Green
Johns Hopkins University
mgreen@cs.jhu.edu

Gabriel Kaptchuk
Johns Hopkins University
gkaptchuk@cs.jhu.edu

Ian Miers
Johns Hopkins University
imiers@cs.jhu.edu

Michael Rushanan
Johns Hopkins University
micharu1@cs.jhu.edu

Abstract

Apple's iMessage is one of the most widely-deployed end-to-end encrypted messaging protocols. Despite its broad deployment, the encryption protocols used by iMessage have never been subjected to rigorous cryptanalysis. In this paper, we conduct a thorough analysis of iMessage to determine the security of the protocol against a variety of attacks. Our analysis shows that iMessage has significant vulnerabilities that can be exploited by a sophisticated attacker. In particular, we outline a novel chosen ciphertext attack on Huffman compressed data, which allows *retrospective* decryption of some iMessage payloads in less than 2^{18} queries. The practical implication of these attacks is that any party who gains access to iMessage ciphertexts may potentially decrypt them remotely and after the fact. We additionally describe mitigations that will prevent these attacks on the protocol, without breaking backwards compatibility. Apple has deployed our mitigations in the latest iOS and OS X releases.

1 Introduction

The past several years have seen widespread adoption of end-to-end encrypted text messaging protocols. In this work we focus on one of the most popular such protocols: Apple's iMessage. Introduced in 2011, iMessage is an end-to-end encrypted text messaging system that supports both iOS and OS X devices. While Apple does not provide up-to-date statistics on iMessage usage, in February 2016 an Apple executive noted that the system had a peak transmission rate of more then 200,000 messages per second, across 1 billion deployed devices [12].

The broad adoption of iMessage has been controversial, particularly within the law enforcement and national security communities. In 2013, the U.S. Drug Enforcement Agency deemed iMessage "a challenge for DEA intercept" [22], while in 2015 the U.S. Department of Justice accused Apple of thwarting an investigation by refusing to turn over iMessage plaintext [11]. iMessage has been at the center of a months-long debate initiated by U.S. and overseas officials over the implementation of "exceptional access" mechanisms in end-to-end encrypted communication systems [7, 26, 33], and some national ISPs have temporarily blocked the protocol [32]. Throughout this controversy, Apple has consistently maintained that iMessage encryption is end-to-end and that even Apple cannot recover the plaintext for messages transmitted through its servers [10].

Given iMessage's large installed base and the high stakes riding on its confidentiality, one might expect iMessage to have received critical attention from the research community. Surprisingly, there has been very little analysis of the system, in large part due to the fact that Apple has declined to publish the details of iMessage's encryption protocol. In this paper we aim to remedy this situation. Specifically, we attempt to answer the following question: how secure is Apple iMessage?

Our contributions. In this work we analyze the iMessage protocol and identify several weaknesses that an attacker may use to decrypt iMessages and attachments. While these flaws do not render iMessage completely insecure, some flaws reduce the level of security to that of the TLS encryption used to secure communications between end-user devices and Apple's servers. This finding is surprising given the protection claims advertised by Apple [10]. Moreover, we determine that the flaws we detect in iMessage may have implications for other aspects of Apple's ecosystem, as we discuss below.

To perform our analysis, we derived a specification for iMessage by conducting a partial black-box reverse engineering of the protocol as implemented on multiple iOS and OS X devices. Our efforts extend a high-level protocol overview published by Apple [9] and two existing partial reverse-engineering efforts [1, 34]. Armed with a protocol specification, we conducted manual cryptanal-

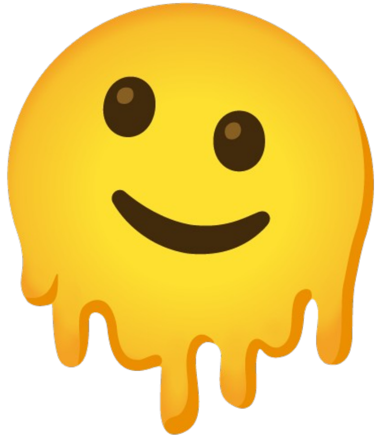
→ replayability

→ malleability

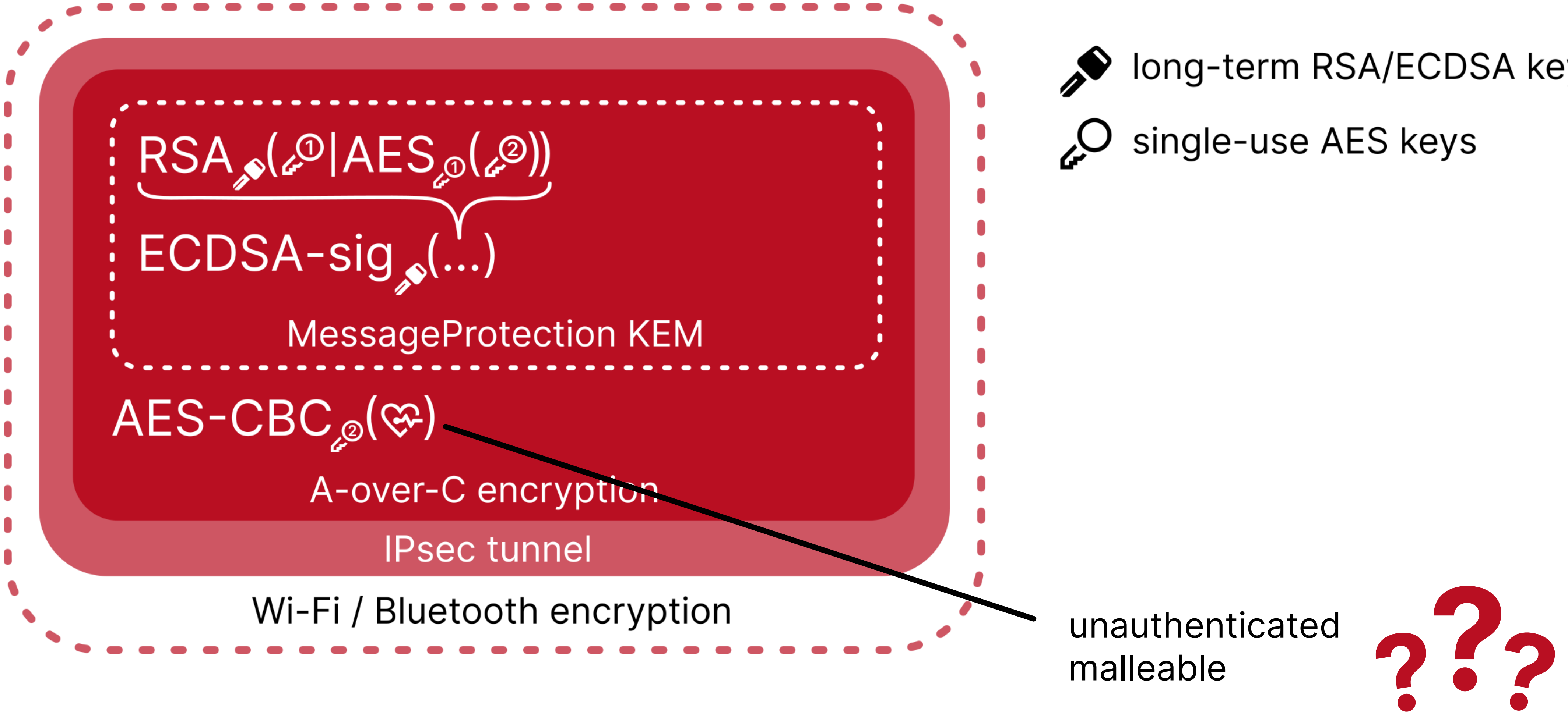
→ no forward secrecy

→ compress-then-encrypt

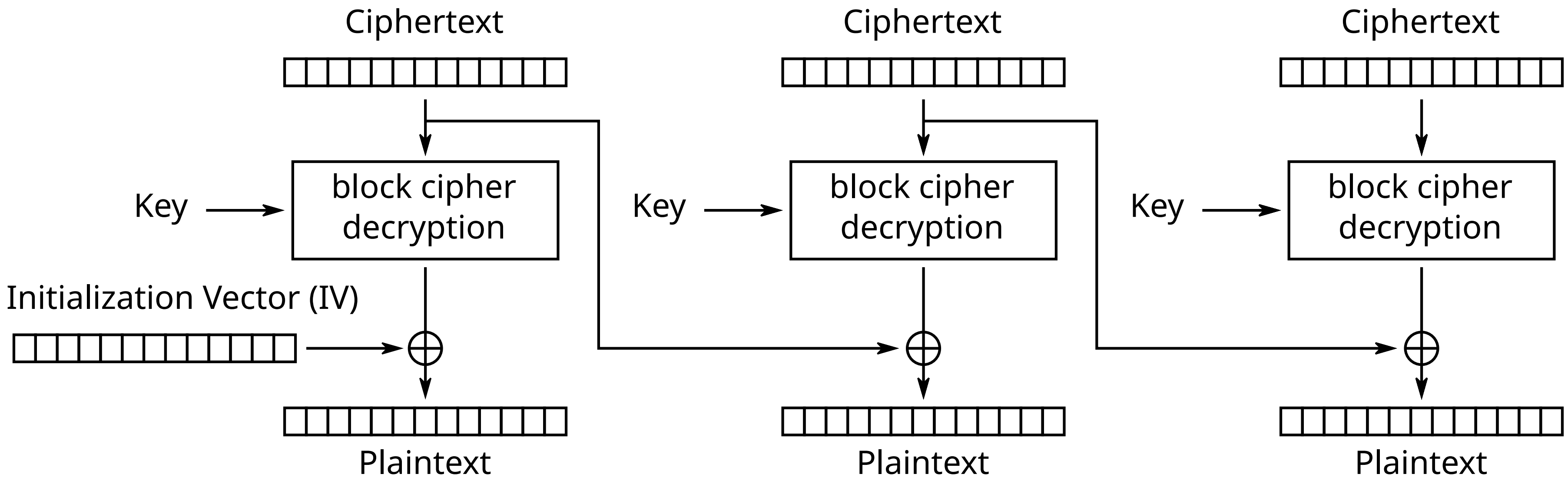
→ weird custom crypto



Malleable Encryption

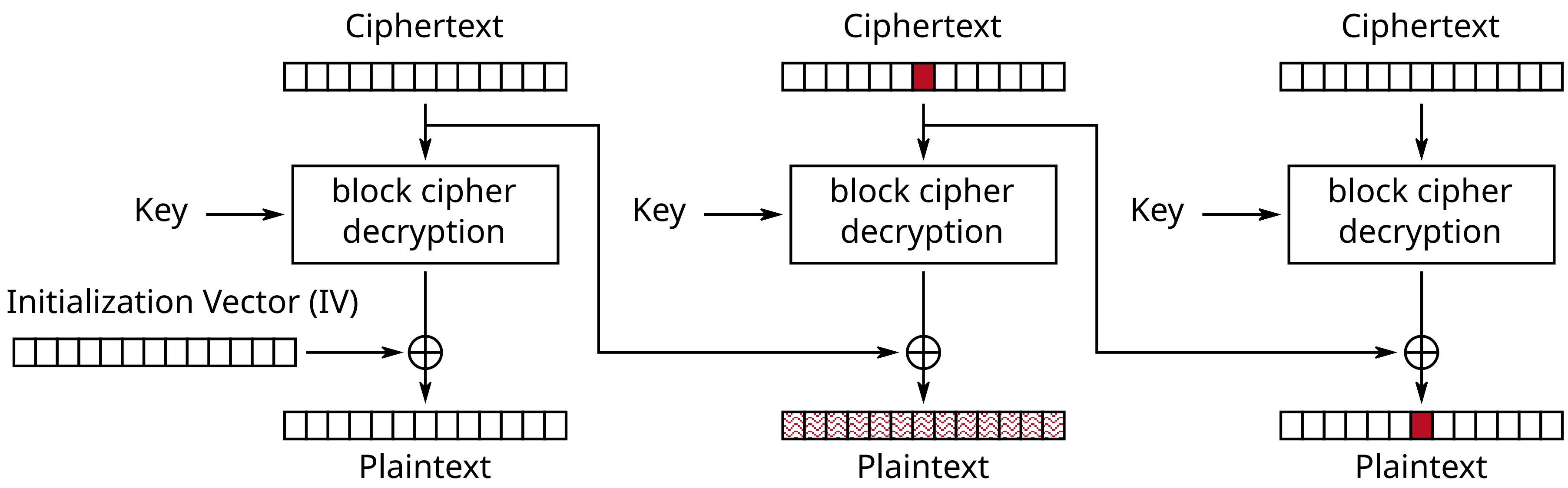


Malleable Encryption



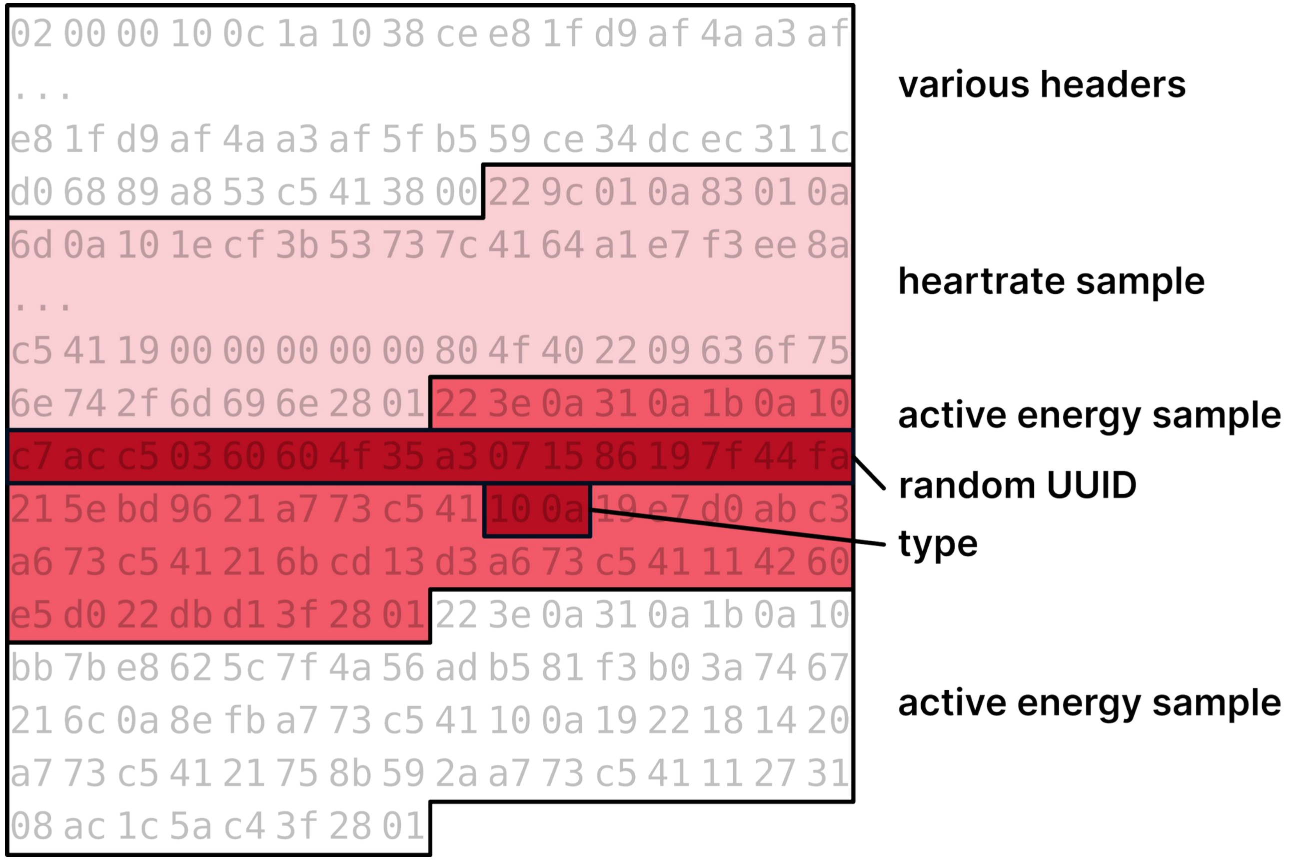
Cipher Block Chaining (CBC) mode decryption

Malleable Encryption



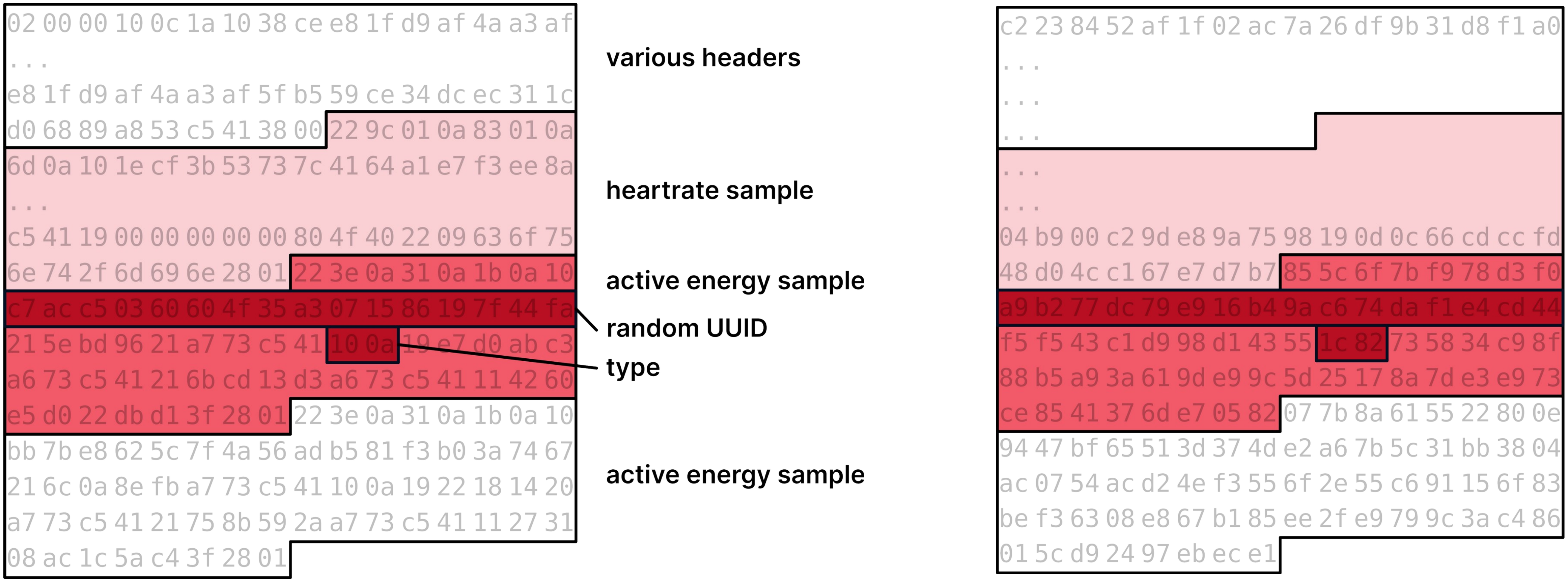
Cipher Block Chaining (CBC) mode decryption

Malleable Encryption



genuine health sync plaintext

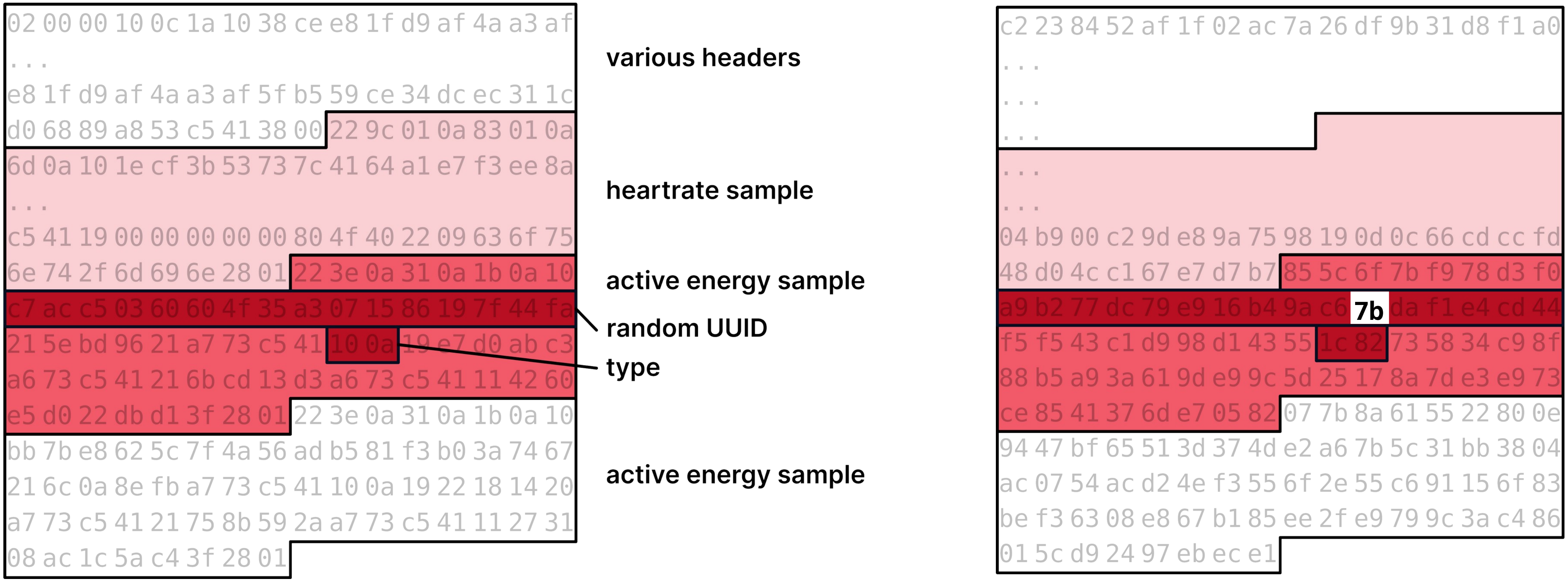
Malleable Encryption



genuine health sync plaintext

A-over-C ciphertext

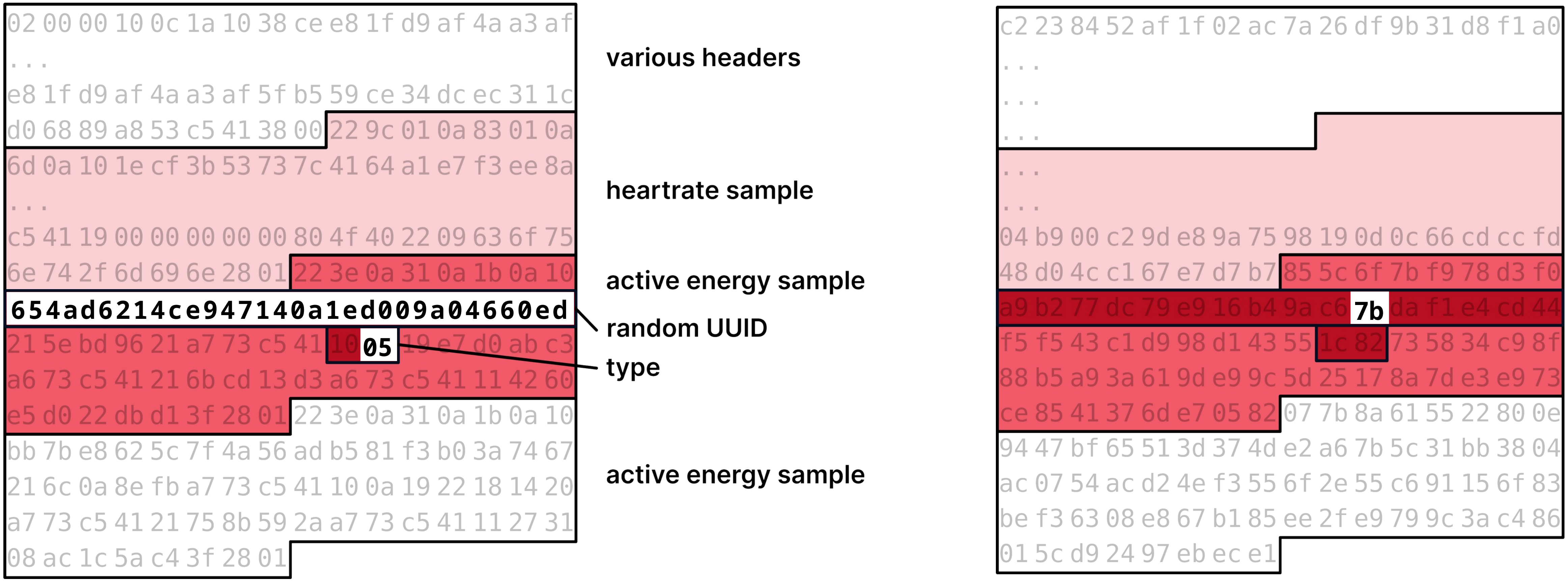
Malleable Encryption



genuine health sync plaintext

A-over-C ciphertext

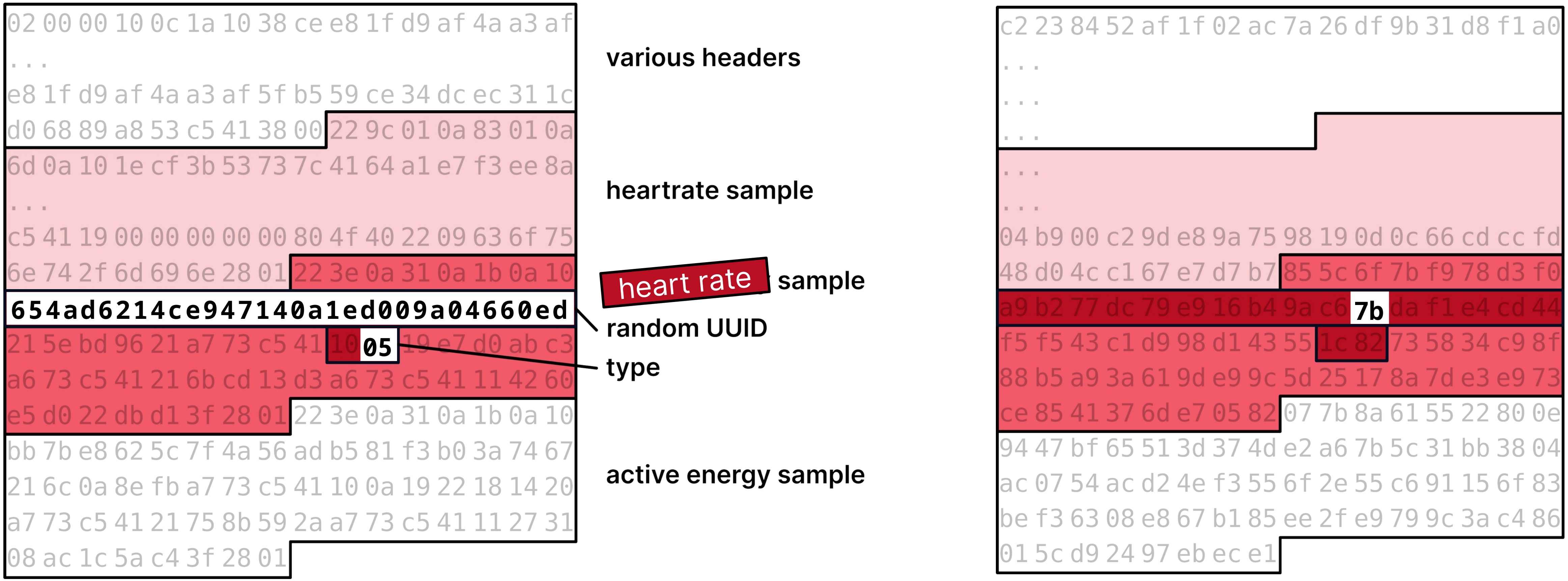
Malleable Encryption



genuine health sync plaintext

A-over-C ciphertext

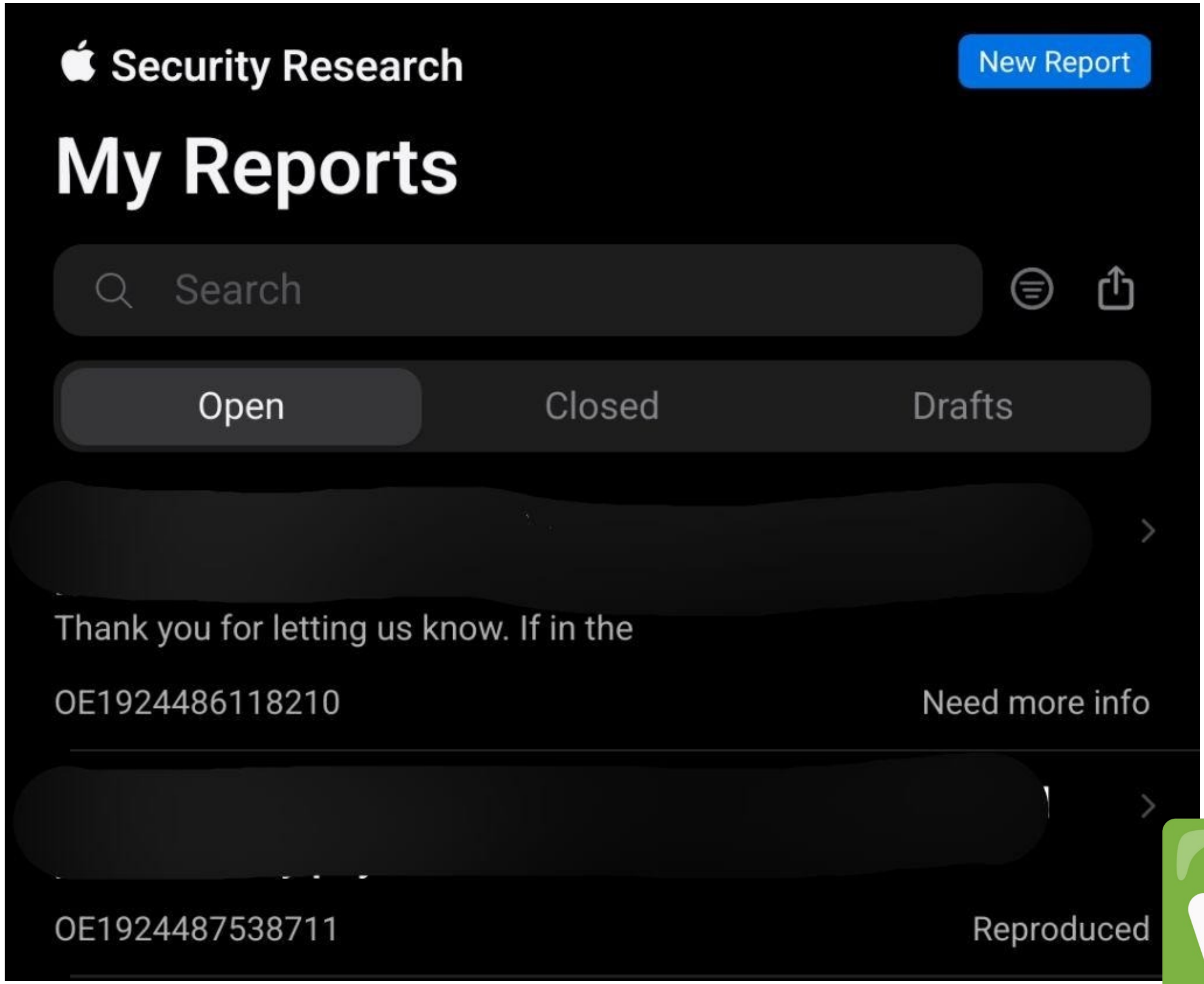
Malleable Encryption



genuine health sync plaintext

A-over-C ciphertext

Responsible Disclosure



Security Takeaways

- Standards exist for a reason
- Crypto will (not) save you
- Consider unexpected system interactions
- Think really really hard before rolling your own crypto
- Avoid complexity whereever possible

DataMessage
AckMessage
KeepAliveMessage
ProtobufMessage
Handshake
EncryptedMessage
DictionaryMessage
AppAckMessage
SessionInvitationMessage
SessionAcceptMessage
SessionDeclineMessage
SessionCancelMessage
SessionMessage
SessionEndMessage
SMSTextMessage
SMSTextDownloadMessage
SMSOutgoing

SMSDownloadOutgoing
SMSDeliveryReceipt
SMSReadReceipt
SMSFailure
FragmentedMessage
ResourceTransferMessage
OTREncryptedMessage
OTRMessage
ProxyOutgoingNiceMessage
ProxyIncomingNiceMessage
TextMessage
DeliveryReceipt
ReadReceipt
AttachmentMessage
PlayedReceipt
SavedReceipt
ReflectedDeliveryReceipt

GenericCommandMessage
GenericGroupMessageCommand
LocationShareOfferCommand
ExpiredAckMessage
ErrorMessage
ServiceMapMessage
SessionReinitiateMessage
SyndicationAction
RetractMessage
EditMessage
RecoverSyncMessage
MarkAsUnreadMessage
DeliveredQuietlyMessage
NotifyRecipientMessage
RecoverJunkMessage
SMSFilteringSettingsMessage

~230 Alloy topics across 151 iOS binaries

accessibility.local accessibility.switchcontrol accounts.representative accountssync addressbooksync airtr
timers amsaccountsync anisette appconduit appconduit.v2 applepay applepay.identitycredential applepay.shar
apppredictionsync appregistrysync appstore appsyncconduit appsyncconduit.v2 arcade.fastsync askto audiocon
audiocontrol.music autobugcapture avconference.avctester biz bluetooth.audio bluetoothregistry bluetoothre
bluetoothregistryclassa bluetoothregistryclassc brook bulletinboard bulletindistributor bulletindistributo
callhistorysync camera.proxy carmelsync carousel.uitrigger clockface.sync cmsession companion.auth compani
contextsync contextsync.local continuity.activity continuity.auth continuity.auth.classa continuity.encryp
continuity.tethering continuity.unlock coreduet coreduet.sync ct.baseband.p2p.notification ct.commcenter.p
ct.commcenter.sim ct.commcenter.sim.cloud ded ded.watch digitalhealth donotdisturb dropin.communication dr
electrictouch eventkitmutation eventkitsync facetime.audio facetime.lp facetime.messaging facetime.multi f
facetime.sync facetime.video familycontrols fignero findmy.itemsharing-crossaccount findmydeviced.aoverc
findmydeviced.watch fi … fmd fmd.local fmf fmf.
gamecenter gelato gfta … ol.cloud groupRemoteCo
groupRemoteControl.ses … haringsetup healthapp.
healthappnotifications … emoteurlconnection ids
intercom internal.watc … manager itunes itunesc
kbd.transfer kcsharing … .auth location.fenceha
location.motion locati … ation.wifitilesync loc
mail.fetches mail.sync … cted mail.sync.protect
maps.eta maps.proxy maps.sync mediaremote mediaremote.v2 messagenotification messages messagesquickswitch
mobiletimersync multiplex1 nameandphoto nanobackup nanomediasync nearby news notes nsurlsessionproxy octag
otaupdate.cloud pairedunlock passbook.general passbook.maintenance passbook.provisioning passbook.relevanc
passbook.remoteadmin pbbridge pbbridge.connectivity pcskey.sync phone.auth phonecontinuity photos.proxy ph
preferencessync preferencessync.pairedsync proxiedcrashcopier proxiedcrashcopier.icloud pushproxy quickboa
regulatorysync remotemediaservices resourcegrabber safari.groupactivities safetymonitor safetymonitor.owna
screenshotter screentime screentimelocal sensorkit sensorkitkeys sessionkit sharing.paireddevice sharing.r
siri.device siri.icloud siri.location siri.phrasespotter siri.proxy siri.voiceshortcuts sleep.classd sms s
sockpuppet sockpuppet.classd soscoordination status.keysharing status.personal suggestions.smartreplies sy
systemsettings systemsettings.files tccd.msg tccd.sync telephonyutilitiestemporary thumper.keys thumper.se
timezonesync tinker.messages tinker.nanoregistry tinker.photos tinker.school tinker.telephony tips usagetr

4. Enter the Witch

PHILIP R. SELLINGER
United States Attorney
By: J. ANDREW RUYMAN
Assistant U.S. Attorney
402 East State Street, Room 430
Trenton, NJ 08608
Telephone: 609-989-0563

JONATHAN S. KANTER
Assistant Attorney General
DOHA G. MEKKI
Principal Deputy Assistant Attorney General
HETAL J. DOSHI
MICHAEL B. KADES
Deputy Assistants Attorney General
By: JONATHAN LASKEN
Assistant Chief, Civil Conduct Task Force
United States Department of Justice
Antitrust Division
450 Fifth Street NW, Suite 8600
Washington, DC 20530
Telephone: 202-598-6517

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF NEW JERSEY

UNITED STATES OF AMERICA U.S. Department of Justice, Antitrust Division 450 Fifth Street NW, Suite 8600 Washington, DC 20530 **STATE OF NEW JERSEY** 124 Halsey Street, 5th Floor Newark, NJ 07102 **STATE OF ARIZONA** 2005 N. Central Avenue Phoenix, AZ 85004 **STATE OF CALIFORNIA** 455 Golden Gate Avenue, Suite 11000 San Francisco, CA 94102 **DISTRICT OF COLUMBIA** 400 6th Street NW, 10th Floor Washington, DC 20001 **STATE OF CONNECTICUT** 165 Capitol Avenue Hartford, CT 06106 **STATE OF MAINE** 6 State House Station	No.

Case 2:24-cv-04055 Document 1 Filed 03/21/

PHILIP R. SELLINGER
United States Attorney
By: J. ANDREW RUYMAN
Assistant U.S. Attorney
402 East State Street, Room 430
Trenton, NJ 08608
Telephone: 609-989-0563

JONA
Assista
DOHA
Princi
HETA
MICH
Deput
By: J
Assis
Unite
Antit
450 F
Wash
Tele

IN THE UNITED STATES DIS
FOR THE DISTRICT OF N

UNITED STATES OF AMERICA
U.S. Department of Justice, Antitrust Division
450 Fifth Street NW, Suite 8600
Washington, DC 20530

STATE OF NEW JERSEY
124 Halsey Street, 5th Floor
Newark, NJ 07102

STATE OF ARIZONA
2005 N. Central Avenue
Phoenix, AZ 85004

STATE OF CALIFORNIA
455 Golden Gate Avenue, Suite 11000
San Francisco, CA 94102

DISTRICT OF COLUMBIA
400 6th Street NW, 10th Floor
Washington, DC 20001

STATE OF CONNECTICUT
165 Capitol Avenue
Hartford, CT 06106

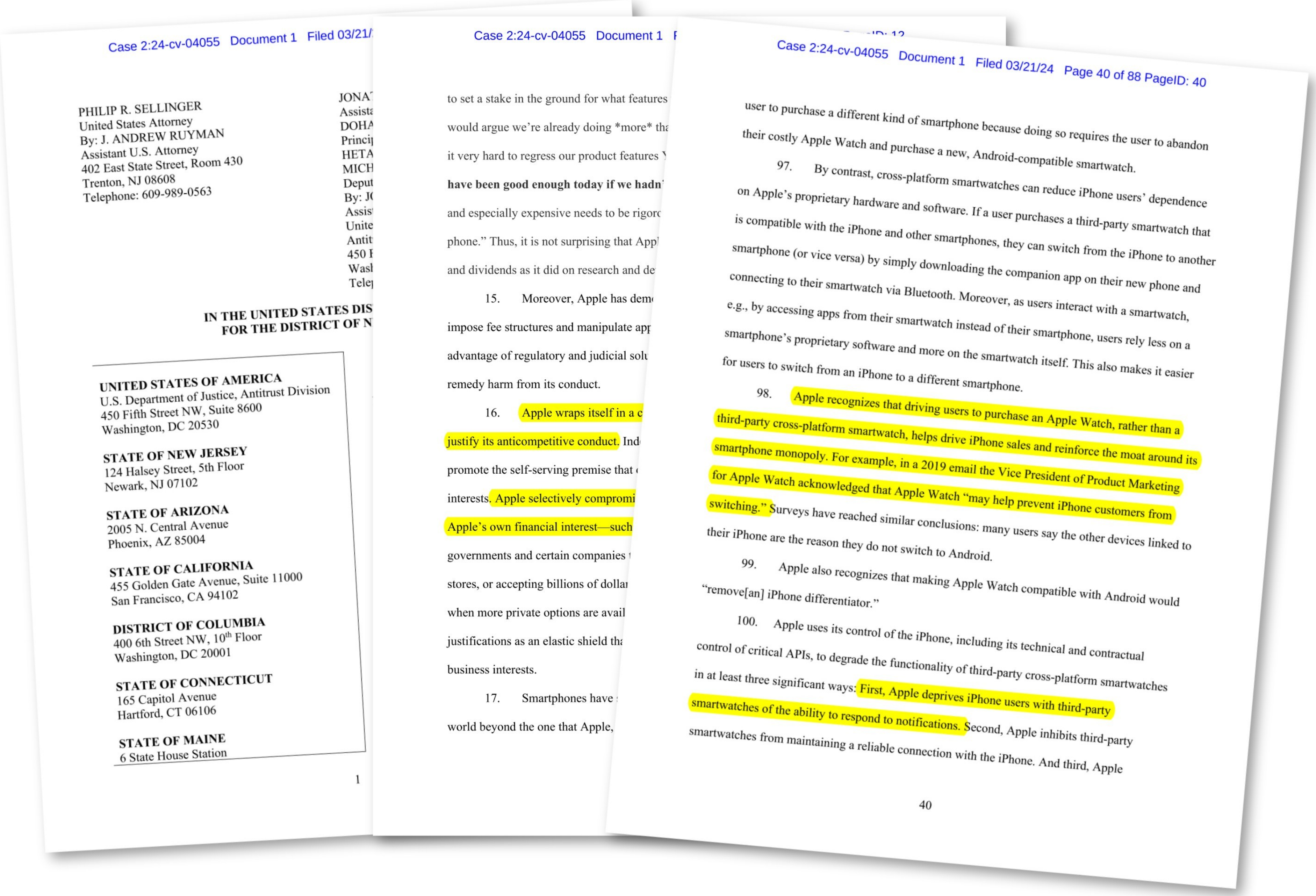
STATE OF MAINE
6 State House Station

to set a stake in the ground for what features we think are 'good enough' for the consumer. I would argue we're already doing *more* than what would have been good enough. But we find it very hard to regress our product features YOY [year over year]." Existing features "**would have been good enough today if we hadn't introduced [them] already**," and "anything new and especially expensive needs to be rigorously challenged before it's allowed into the consumer phone." Thus, it is not surprising that Apple spent more than twice as much on stock buybacks and dividends as it did on research and development.

15. Moreover, Apple has demonstrated its ability to use its smartphone monopoly to impose fee structures and manipulate app review to inhibit aggrieved parties from taking advantage of regulatory and judicial solutions imposed on Apple that attempt to narrowly remedy harm from its conduct.

16. Apple wraps itself in a cloak of privacy, security, and consumer preferences to justify its anticompetitive conduct. Indeed, it spends billions on marketing and branding to promote the self-serving premise that only Apple can safeguard consumers' privacy and security interests. Apple selectively compromises privacy and security interests when doing so is in Apple's own financial interest—such as degrading the security of text messages, offering governments and certain companies the chance to access more private and secure versions of app stores, or accepting billions of dollars each year for choosing Google as its default search engine when more private options are available. In the end, Apple deploys privacy and security justifications as an elastic shield that can stretch or contract to serve Apple's financial and business interests.

17. Smartphones have so revolutionized American life that it can be hard to imagine a world beyond the one that Apple, a self-interested monopolist, deems "good enough." But under


Case 2:24-cv-04055 Document 1 Filed 03/21/

PHILIP R. SELLINGER
United States Attorney
By: J. ANDREW RUYMAN
Assistant U.S. Attorney
402 East State Street, Room 430
Trenton, NJ 08608
Telephone: 609-989-0563

JONA
Assista
DOHA
Princi
HETA
MICH
Deput
By: J
Assis
Unite
Antit
450 F
Wash
Telep

IN THE UNITED STATES DIS
FOR THE DISTRICT OF N

UNITED STATES OF AMERICA
U.S. Department of Justice, Antitrust Division
450 Fifth Street NW, Suite 8600
Washington, DC 20530

STATE OF NEW JERSEY
124 Halsey Street, 5th Floor
Newark, NJ 07102

STATE OF ARIZONA
2005 N. Central Avenue
Phoenix, AZ 85004

STATE OF CALIFORNIA
455 Golden Gate Avenue, Suite 11000
San Francisco, CA 94102

DISTRICT OF COLUMBIA
400 6th Street NW, 10th Floor
Washington, DC 20001

STATE OF CONNECTICUT
165 Capitol Avenue
Hartford, CT 06106

STATE OF MAINE
6 State House Station

1

Case 2:24-cv-04055 Document 1 F

to set a stake in the ground for what features
would argue we're already doing *more* tha
it very hard to regress our product features
have been good enough today if we hadn'
and especially expensive needs to be rigoro
phone." Thus, it is not surprising that Appl
and dividends as it did on research and de

15. Moreover, Apple has dem
impose fee structures and manipulate app
advantage of regulatory and judicial solu
remedy harm from its conduct.

16. Apple wraps itself in a c
justify its anticompetitive conduct. Ind
promote the self-serving premise that
interests. Apple selectively compromi
Apple's own financial interest—such
governments and certain companies
stores, or accepting billions of dolla
when more private options are avail
justifications as an elastic shield tha
business interests.

17. Smartphones have
world beyond the one that Apple,

Case 2:24-cv-04055 Document 1 Filed 03/21/24 Page 40 of 88 PageID: 40

user to purchase a different kind of smartphone because doing so requires the user to abandon their costly Apple Watch and purchase a new, Android-compatible smartwatch.

97. By contrast, cross-platform smartwatches can reduce iPhone users' dependence on Apple's proprietary hardware and software. If a user purchases a third-party smartwatch that is compatible with the iPhone and other smartphones, they can switch from the iPhone to another smartphone (or vice versa) by simply downloading the companion app on their new phone and connecting to their smartwatch via Bluetooth. Moreover, as users interact with a smartwatch, e.g., by accessing apps from their smartwatch instead of their smartphone, users rely less on a smartphone's proprietary software and more on the smartwatch itself. This also makes it easier for users to switch from an iPhone to a different smartphone.

98. Apple recognizes that driving users to purchase an Apple Watch, rather than a third-party cross-platform smartwatch, helps drive iPhone sales and reinforce the moat around its smartphone monopoly. For example, in a 2019 email the Vice President of Product Marketing for Apple Watch acknowledged that Apple Watch "may help prevent iPhone customers from switching." Surveys have reached similar conclusions: many users say the other devices linked to their iPhone are the reason they do not switch to Android.

99. Apple also recognizes that making Apple Watch compatible with Android would "remove[an] iPhone differentiator."

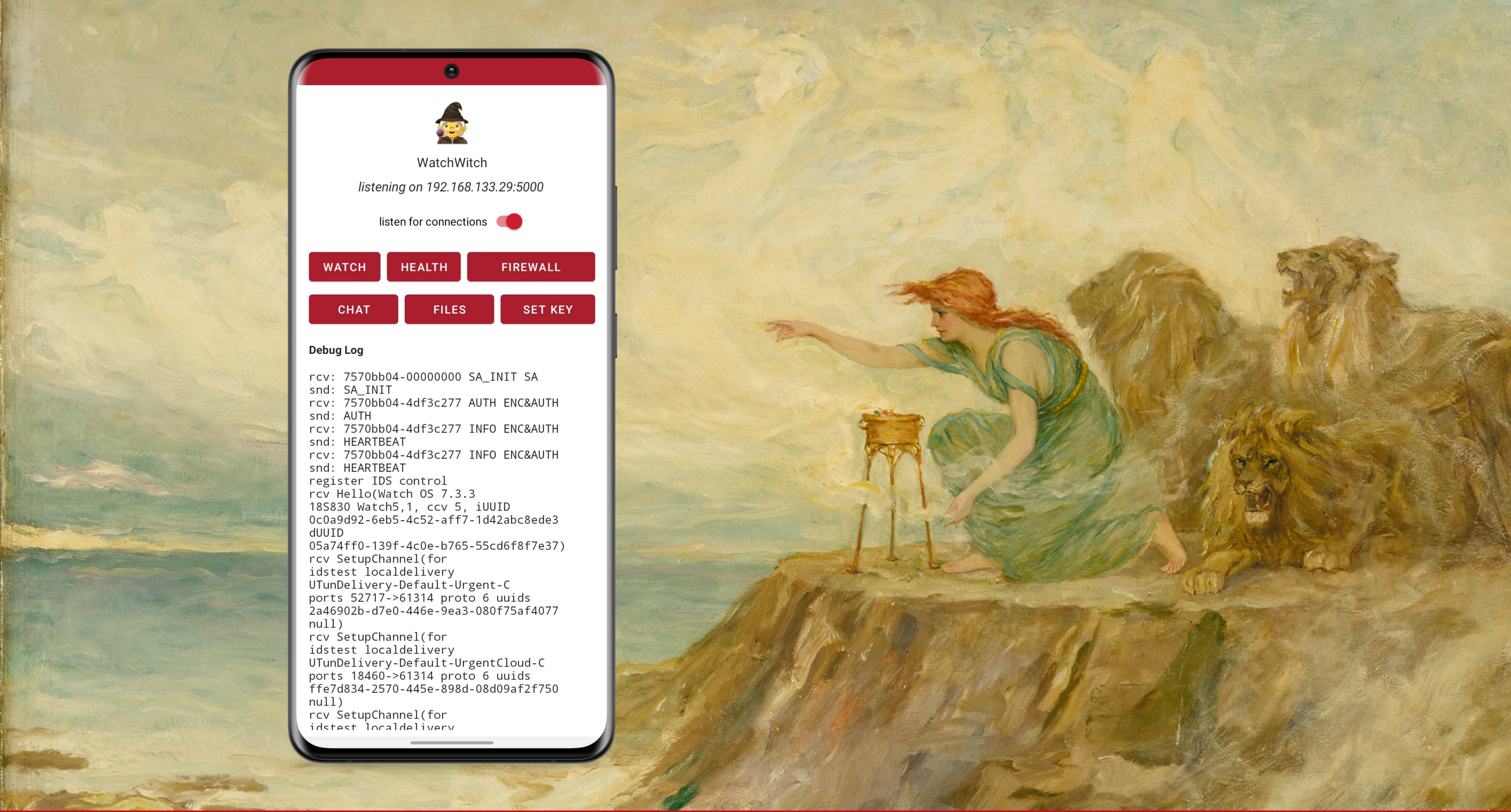
100. Apple uses its control of the iPhone, including its technical and contractual control of critical APIs, to degrade the functionality of third-party cross-platform smartwatches in at least three significant ways: First, Apple deprives iPhone users with third-party smartwatches of the ability to respond to notifications. Second, Apple inhibits third-party smartwatches from maintaining a reliable connection with the iPhone. And third, Apple

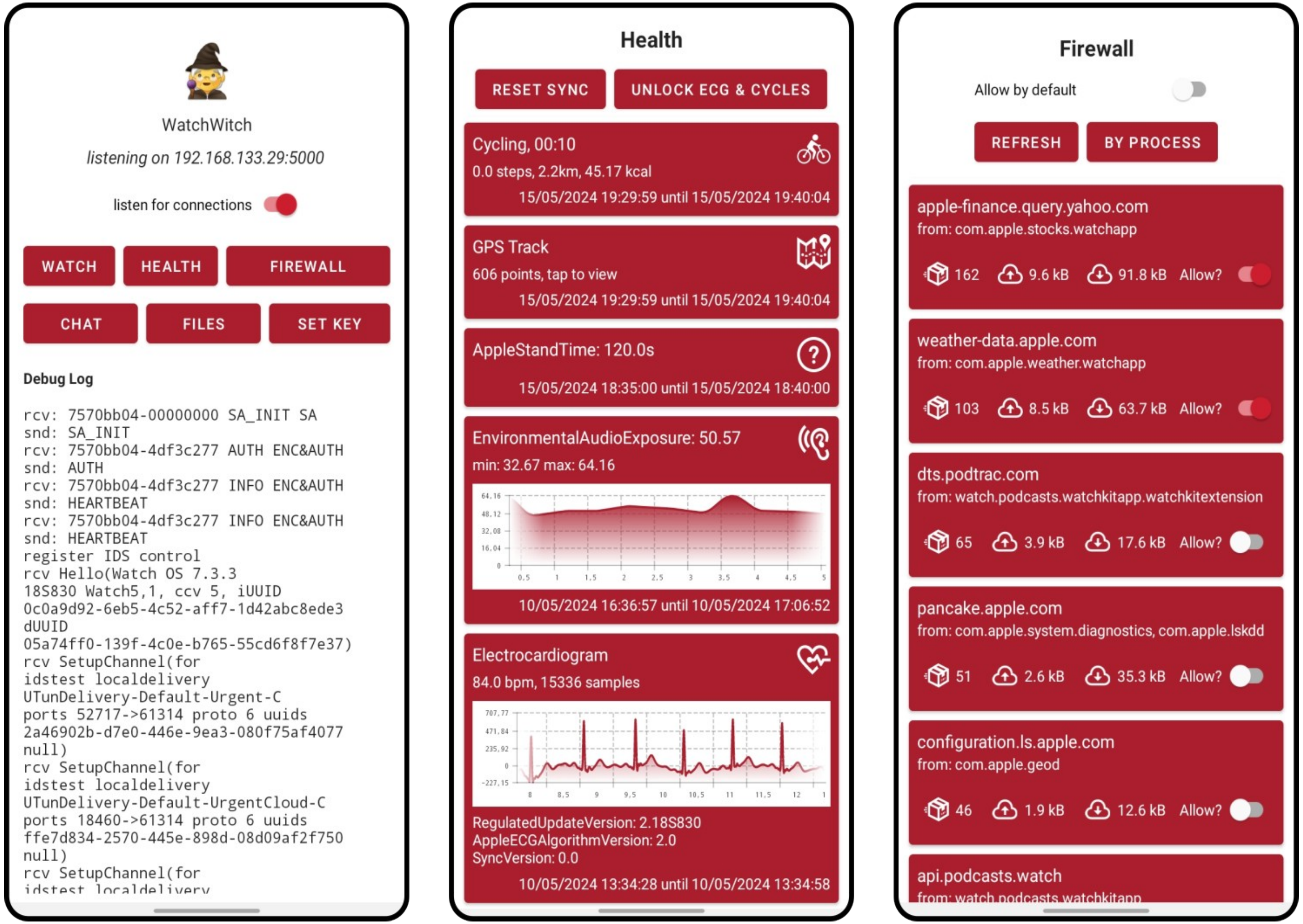
40

Apple says it spent three years trying to bring Apple Watch to Android

Chance Miller | Mar 21 2024 - 12:03 pm PT | 150 Comments

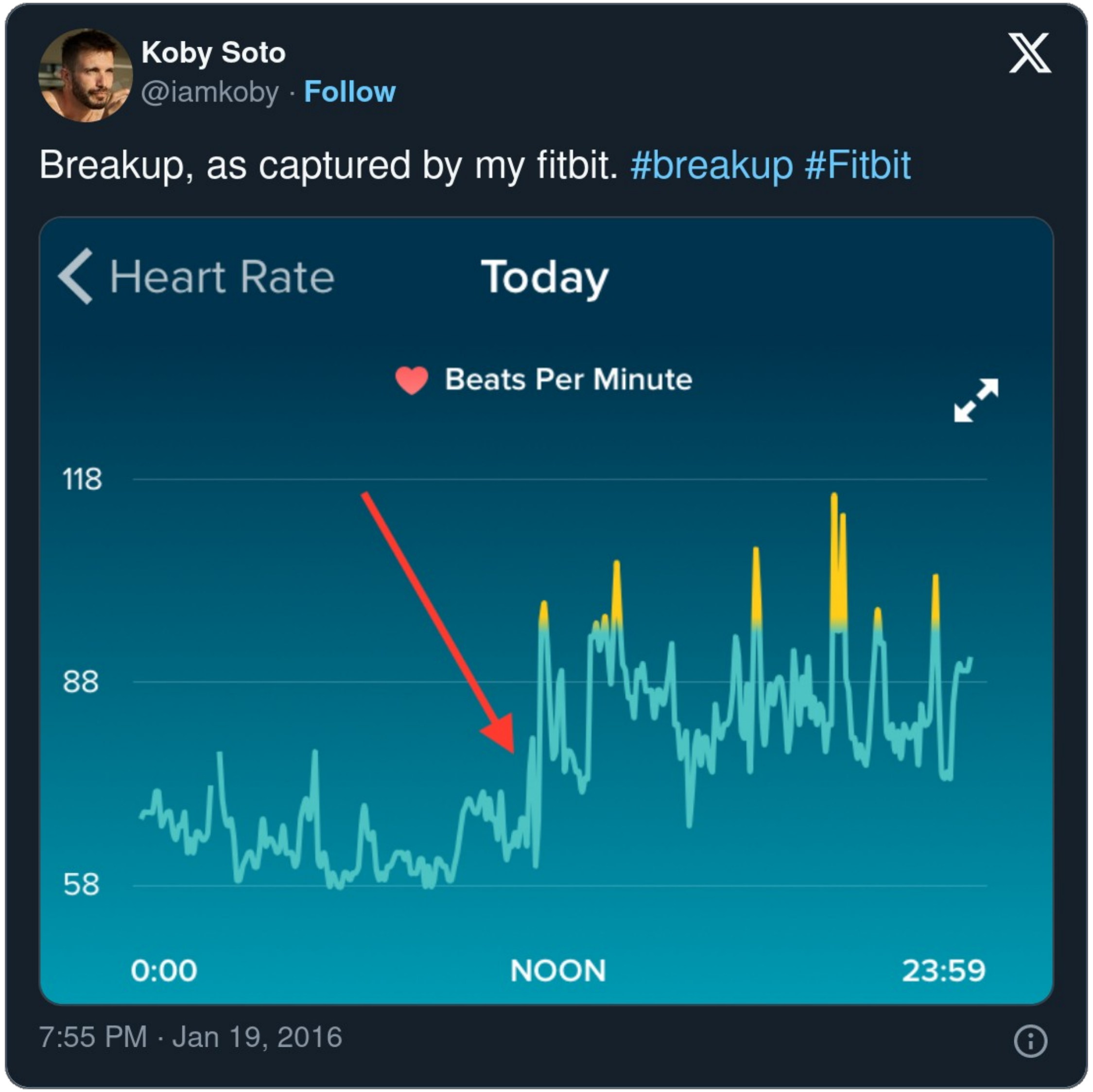
https://9to5mac.com/2024/03/21/apple-watch-android-apple-work/


WatchWitch
listening on 192.168.133.29:5000
listen for connections
WATCH
HEALTH
FIREWALL
CHAT
FILES
SET KEY
Debug Log
rcv: 7570bb04-00000000 SA_INIT SA
snd: SA_INIT
rcv: 7570bb04-4df3c277 AUTH ENC&AUTH
snd: AUTH
rcv: 7570bb04-4df3c277 INFO ENC&AUTH
snd: HEARTBEAT
rcv: 7570bb04-4df3c277 INFO ENC&AUTH
snd: HEARTBEAT
register IDS control
rcv Hello(Watch OS 7.3.3
18S830 Watch5,1, ccv 5, iUUID
0c0a9d92-6eb5-4c52-aff7-1d42abc8ede3
dUUID
05a74ff0-139f-4c0e-b765-55cd6f8f7e37)
rcv SetupChannel(for
idstest localdelivery
UTunDelivery-Default-Urgent-C
ports 52717->61314 proto 6 uuids
2a46902b-d7e0-446e-9ea3-080f75af4077
null)
rcv SetupChannel(for
idstest localdelivery
UTunDelivery-Default-UrgentCloud-C
ports 18460->61314 proto 6 uuids
ffe7d834-2570-445e-898d-08d09af2f750
null)
rcv SetupChannel(for


WatchWitch
listening on 192.168.133.29:5000
listen for connections
WATCH
HEALTH
FIREWALL
CHAT
FILES
SET KEY
Debug Log
rcv: 7570bb04-00000000 SA_INIT SA
snd: SA_INIT
rcv: 7570bb04-4df3c277 AUTH ENC&AUTH
snd: AUTH
rcv: 7570bb04-4df3c277 INFO ENC&AUTH
snd: HEARTBEAT
rcv: 7570bb04-4df3c277 INFO ENC&AUTH
snd: HEARTBEAT
register IDS control
rcv Hello(Watch OS 7.3.3
18S830 Watch5,1, ccv 5, iUUID
0c0a9d92-6eb5-4c52-aff7-1d42abc8ede3
dUUID
05a74ff0-139f-4c0e-b765-55cd6f8f7e37)
rcv SetupChannel(for
idstest localdelivery
UTunDelivery-Default-Urgent-C
ports 52717->61314 proto 6 uuids
2a46902b-d7e0-446e-9ea3-080f75af4077
null)
rcv SetupChannel(for
idstest localdelivery
UTunDelivery-Default-UrgentCloud-C
ports 18460->61314 proto 6 uuids
ffe7d834-2570-445e-898d-08d09af2f750
null)
rcv SetupChannel(for
Health
RESET SYNC
UNLOCK ECG & CYCLES
Cycling, 00:10
0.0 steps, 2.2km, 45.17 kcal
15/05/2024 19:29:59 until 15/05/2024 19:40:04
GPS Track
606 points, tap to view
15/05/2024 19:29:59 until 15/05/2024 19:40:04
AppleStandTime: 120.0s
15/05/2024 18:35:00 until 15/05/2024 18:40:00
EnvironmentalAudioExposure: 50.57
min: 32.67 max: 64.16
10/05/2024 16:36:57 until 10/05/2024 17:06:52
Electrocardiogram
84.0 bpm, 15336 samples
RegulatedUpdateVersion: 2.18S830
AppleECGAlgorithmVersion: 2.0
SyncVersion: 0.0
10/05/2024 13:34:28 until 10/05/2024 13:34:58
Firewall
Allow by default
REFRESH
BY PROCESS
apple-finance.query.yahoo.com
from: com.apple.stocks.watchapp
162 9.6 kB 91.8 kB Allow?
weather-data.apple.com
from: com.apple.weather.watchapp
103 8.5 kB 63.7 kB Allow?
dts.podtrac.com
from: watch.podcasts.watchkitapp.watchkitextension
65 3.9 kB 17.6 kB Allow?
pancake.apple.com
from: com.apple.system.diagnostics, com.apple.lskdd
51 2.6 kB 35.3 kB Allow?
configuration.ls.apple.com
from: com.apple.geod
46 1.9 kB 12.6 kB Allow?
api.podcasts.watch

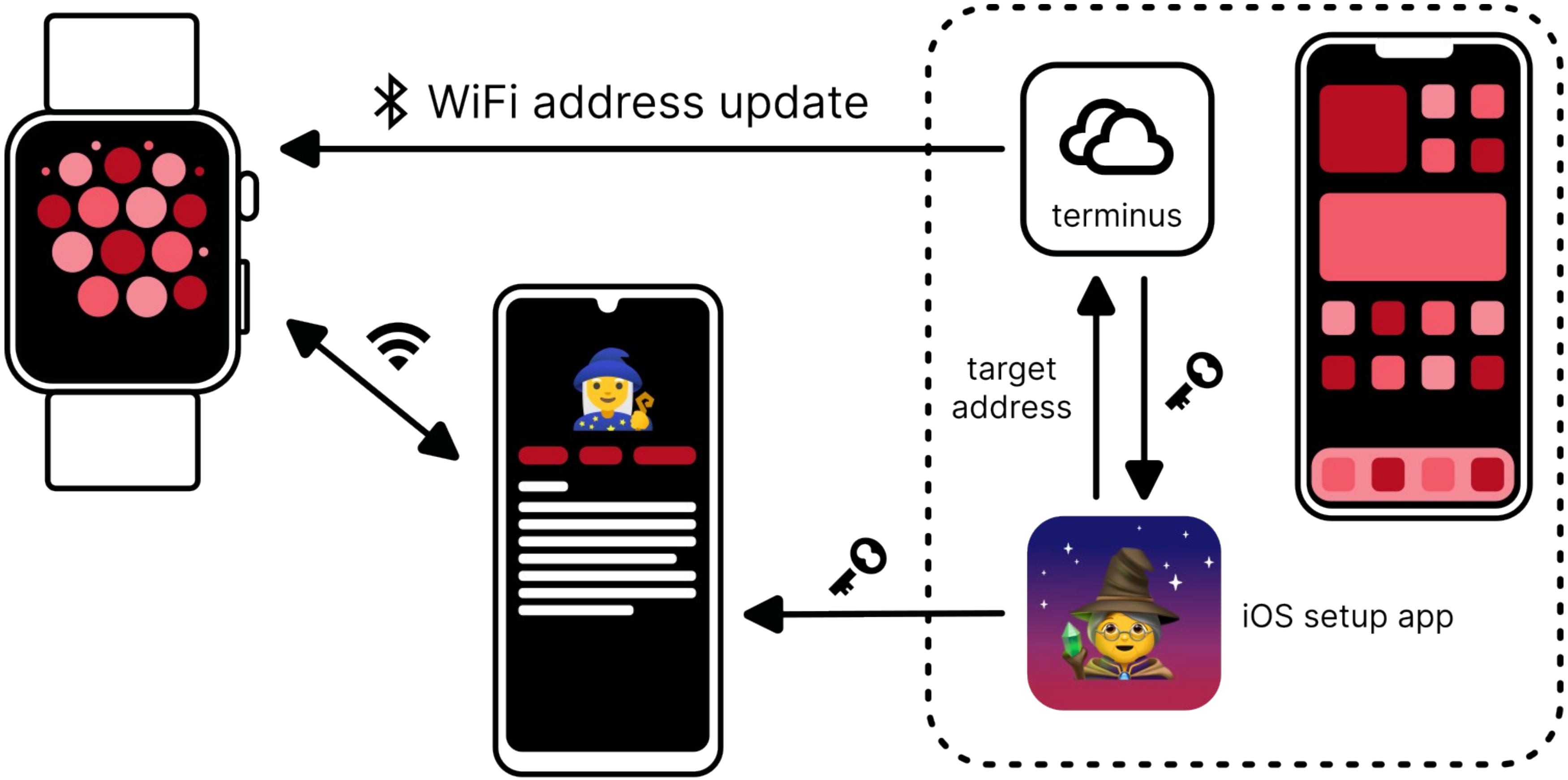

Medea
Verified
Wed, 15 May
the moon is so big tonight 17:31
17:31
Signal message


17:31
SIGNAL
Circe
the moon is so big tonight
Reply

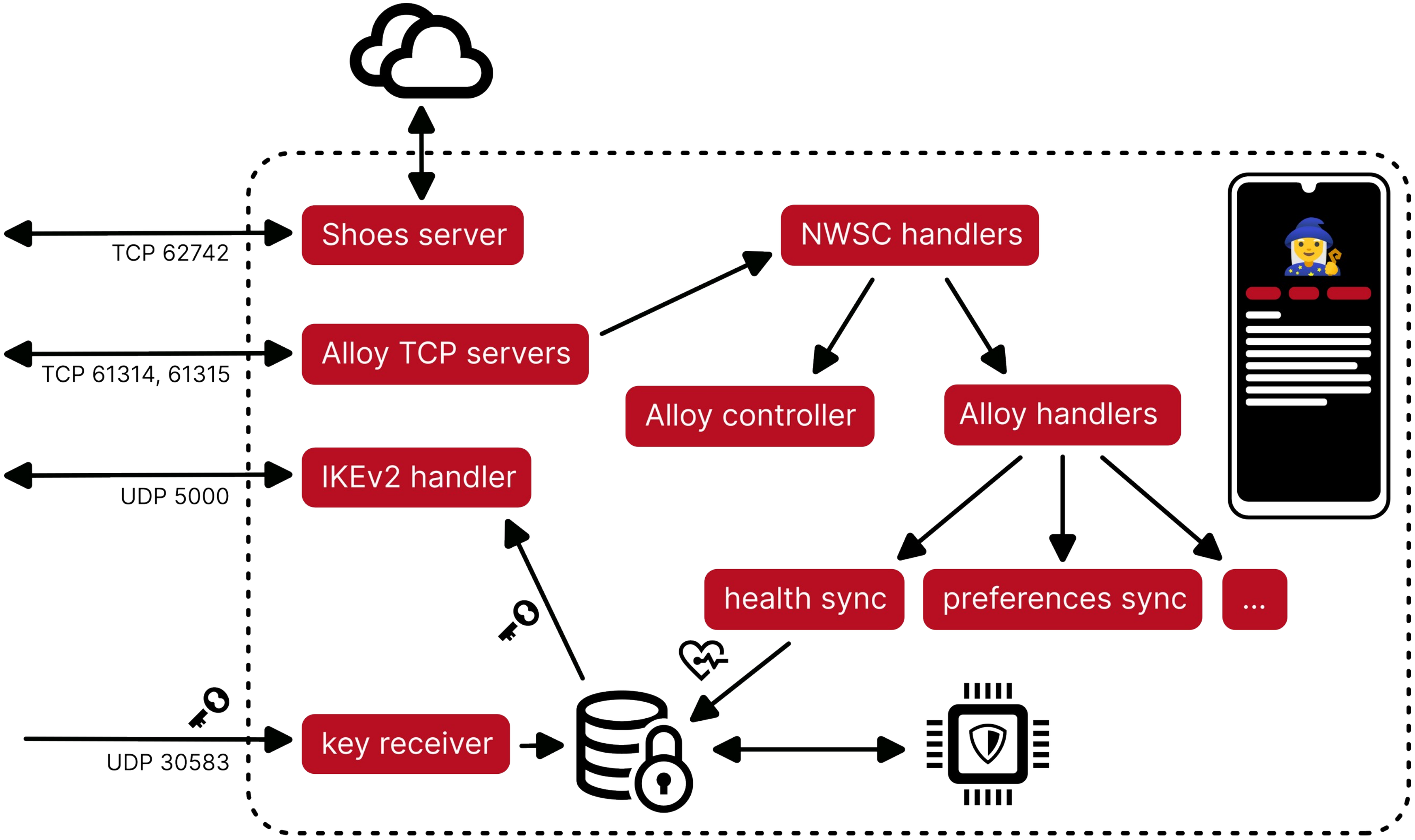

Koby Soto
@iamkoby · Follow
Breakup, as captured by my fitbit. #breakup #Fitbit
Heart Rate
Today
Beats Per Minute
118
88
58
0:00
NOON
23:59
7:55 PM · Jan 19, 2016


Koby Soto
@iamkoby · Follow
Troopers, as captured by my Apple Watch.
Demo? 👀
7:55 PM · Jan 19, 2016

App Architecture



App Architecture



Interoperability Takeaways

- It can be done.

- It can be secure.

- Open Interfaces are curb cuts

- A better world is possible ✨

app release & source code

tooling, frida scripts, wireshark dissectors

more protocol documentation

stay tuned ✨

5. One more thing...

bytewitch

bplist protobuf opack nsarchive generic

0a305369676e616c2e4170704e6f74696669636174696f6e732e416374696f6e2e7265616374576974685468756d6273557012080a04f09f918d180018012800

live decode

decode try harder

protobuf

1	"Signal.AppNotifications.Action.reactWithThumbsUp"
2	1 "👍" 3 VarInt(0)
3	VarInt(1)
5	VarInt(0)

bytewitch

bplist | protobuf | opack | nsarchive | generic

01328e0162706c6973743030d2010203045f1028554e4e6f74696669636174696f6e416374696f6e54657874496e707574506c6163656
86f6c6465725f1028554e4e6f74696669636174696f6e416374696f6e54657874496e707574427574746f6e5469746c65505453656e64
080d3863640000000000000001010000000000000000050000000000000000000000000000000069

☑ live decode

decode | try harder

protobuf

2	"org.whispersystems.signal"
6	"💚 Sticker Message"
7	I64(unix time Mon Mar 25 2024 14:46:55 GMT+0100 (Central European Standard Time))
9	1: "Signal.AppNotifications.Action.reply" 2: 1: "Reply"; 5: VarInt(1) 6: "UNNotificationActionTextInputPlaceholder" → ""; "UNNotificationActionTextInputButtonTitle" → "Send"

SEMC

bytewitch

bplist protobuf opack nsarchive generic

70049004c00510053006700 6d0074007c0087008e009300950097009 9009b00a000a200a400a600a800aa00b600c100dc00e500e600eb
00f300fc00fe0103010e0117011e0121012a012b013201350137013913c013e014001420171019f01c601ed01f201ff02020207021d0
0000000000002010000000000000003d0000000000000000000000000000221

live decode

decode try harder

protobuf

2	"org.whispersystems.signal"
9	1 "Signal.AppNotifications.Action.reply" 2 — 1 "Reply"; 5 VarInt(1) 6 — "UNNotificationActionTextInputPlaceholder" → ""; "UNNotificationActionTextInputButtonTitle" → "Send"
21	"launchImage" → "" "recordDate" → Mon Mar 25 2024 14:46:55 GMT+0100 (Central European Standard Time) "shouldIgnoreDoNotDisturb" → false "userInfo" → "Signal.AppNotificationsUserInfoKey.messageId" → "C13FEA11-04BE-40EB-87EB-235034DA2FE4"; "Signal.AppNotificationsUserInfoKey.threadId" → "163CD1F9-5415-49C5-8C61-869D4000AF15"

bytewitch

rec0de.net/open/bytewitch/

github.com/rec0de/bytewitch/

✨Questions✨

nrollshausen@seemoo.de

@trusted_device@infosec.exchange