black hat BRIEFINGS

AUGUST 6-7, 2025

MANDALAY BAY / LAS VEGAS

Not Sealed:

Practical Attacks on Nostr, a Decentralized Censorship-Resistant Protocol

Keywords: Distributed SNS, signature verification bypass, CBC mode malleability, cache poisoning, plaintext recovery

Speakers: Hayato Kimura

Contributors: Ryoma Ito, Kazuhiko Minematsu, Shogo Shiraki and Takanori Isobe

(Also, IEEE EuroS&P2025)

Our Team

Hayato Kimura

- Researcher at NICT, Japan
  (**N**ational **I**nstitute of information and **C**ommunications **T**echnology)
- Ph.D. candidate at The University of Osaka
- Research field: Applied Cryptography & Protocol Security

Ryoma Ito
(NICT)

Kazuhiko Minematsu
(NEC)

Shogo Shiraki
(University of Hyogo)

Takanori Isobe
(The University of Osaka)

The dawn of the Distributed SNS

FORBES DIGITAL ASSETS

Jack Dorsey Backs Ocean In Shifting Toward Decentralized Bitcoin Mining

By Susie Violet Ward, Contributor. Bitcoin journalist and financial analyst b... Follow Author

Published Dec 01, 2023, 03:56am EST

BITCOIN

BITCOIN MAGAZINE

Why Nostr Today Feels Like Bitcoin In 2012: An Interview With Vitor Pamplona

May 16, 2025 — 04:56 pm EDT

Written by Frank Corva for Bitcoin Magazine →

SOCIAL

Jack Dorsey pumps $10M into a nonprofit focused on open source social media

Sarah Perez — 9:01 PM PDT · July 16, 2025

Mastodon's Growth, and Communities Branching

April 30, 2023 / Hilda Bastian / Science Communication

Meet @Fiatjaf, The Myster Nostr Creator Who Has Lu 18 Million Users And $5 Million From Jack Dorsey

Threads Surpasses 350M Monthly Users: A Growth Milestone with Mixed Monetization Prospects

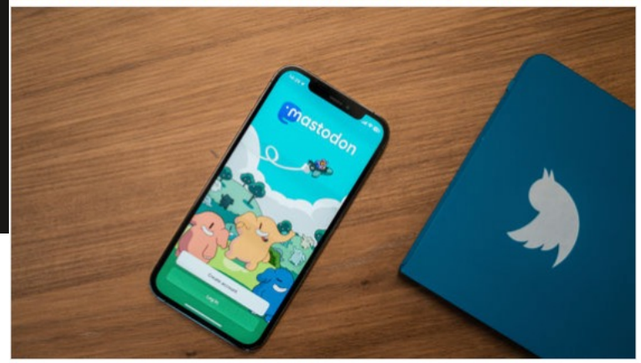
Nathaniel Stone • Wednesday, Apr 30, 2025 6:54 pm ET 4min read

Bluesky adds 1m new members as users flee X after the US election

Social media platform has become a 'refuge' from the far-right activism on X, experts say, after Elon Musk teamed up with Donald Trump

Feb 10, 2023 19:00:00

Is distributed SNS, which has grown rapidly after Earon Mask's acquisition of Twitter, sluggish?



Distributed SNS

Self-sovereign

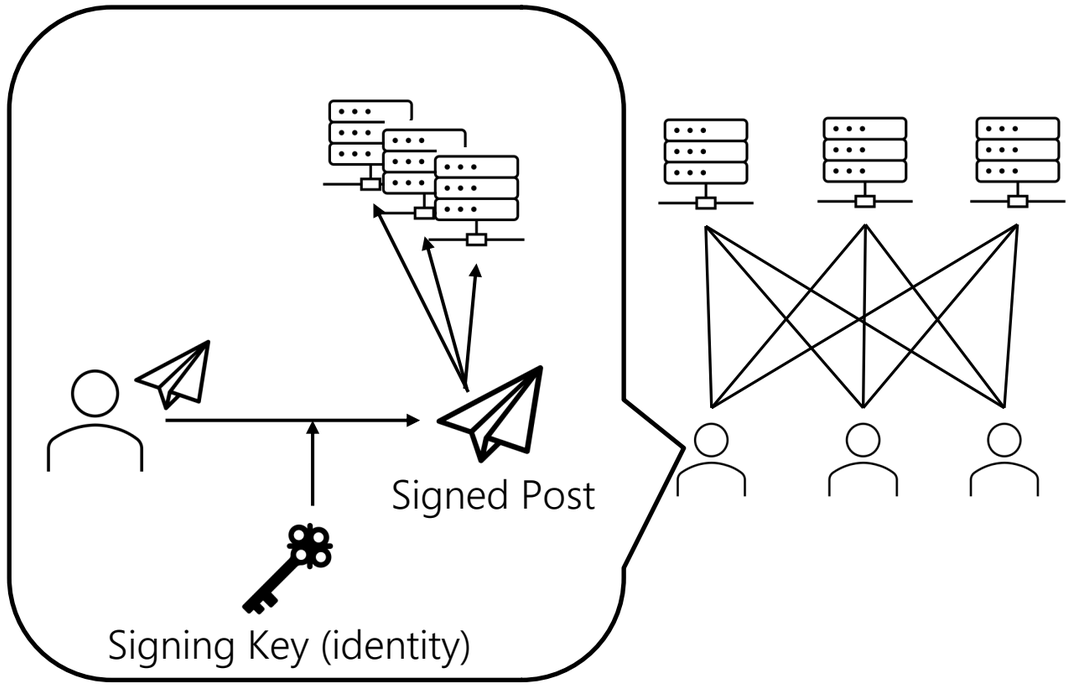
NOSTR

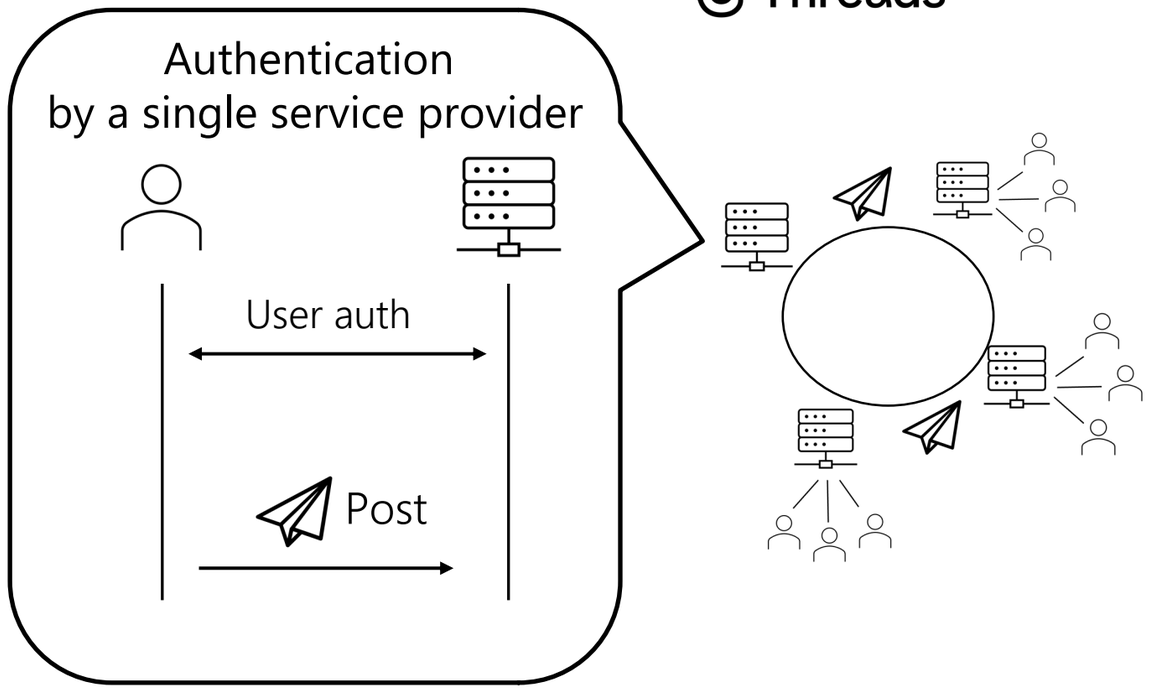
Bluesky

Federated

mastodon Misskey

Threads





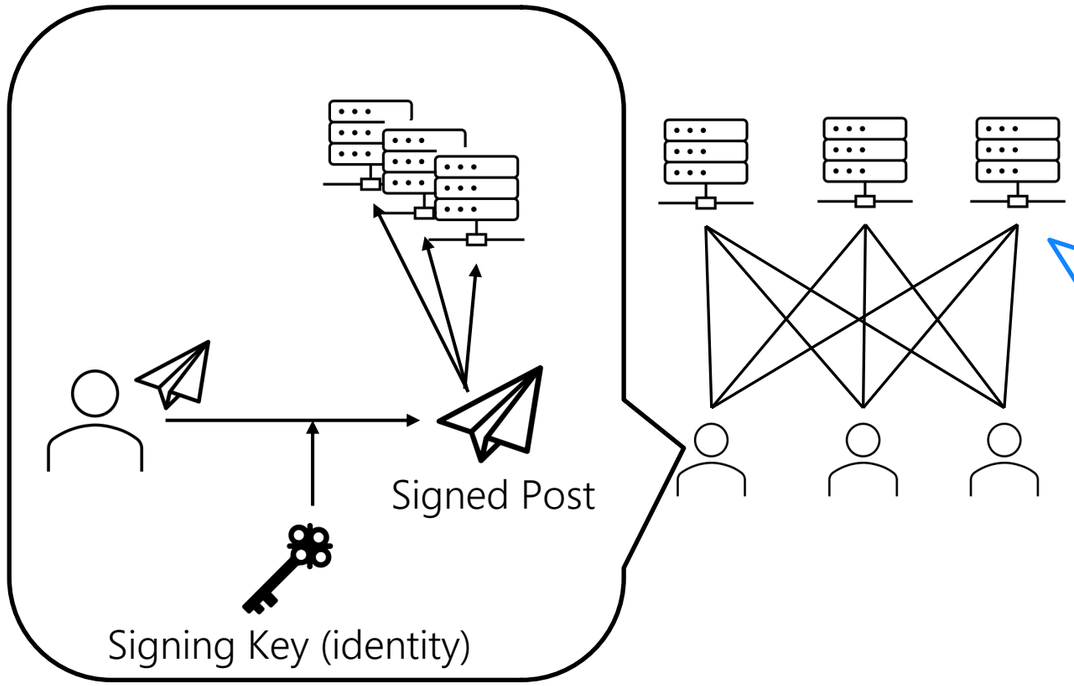
Service providers are independent
User's identity is managed by user

Service providers are interconnected
But identity managed like a **centralized** SNS

Distributed SNS

Self-sovereign

NOSTR



Service providers are independent
User's identity is managed by user

Blue

Quite different architecture from traditional centralized SNS / messaging

Research Questions

- How to trust public keys?
- New architecture, new attack surface?

Service providers are interconnected
But identity managed like a **centralized** SNS

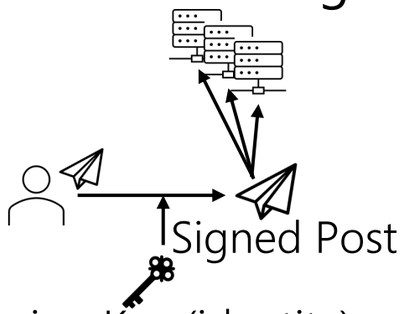
What is Nostr?

- **Open, censorship-resistant social-network**
- **1.1 million registration users**
- **No centralized authority, users must manage Public-key-based identities**
  - A secp256k1 key pair defines who you are; every post carries a signature
- **Zero barriers to participation**
  - Anyone can run a relay server or client
  - Covers most of the attractive features of centralized SNS
    - E.g., Post, Profile, Encrypted DMs, Micro payment, Multiple device sign-in

Cryptography in Nostr Specs

- NIP = Nostr implementation possibilities
- 56+ specifications
- 1 mandatory protocol & 55+ optional protocols
- 4 key feature protocols

NIP-01:
Event Structure & Signing



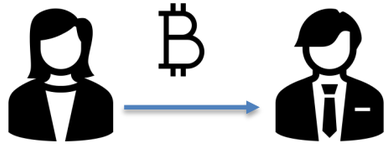
NIP-04:
Encrypted DM

ECDH + AES-CBC
+ Signing

NIP-46:
Delegation (multi-device)

Send Token via
ECDH + AES-CBC
+ Signing

NIP-57:
Micro payment



Our Contributions

- Analyze 56 specs
- Analyze 9 implementations
- Find 7 vulnerabilities on 4 key features

- Implement 8 attacks
- Breaking confidentiality, integrity, availability

- Propose mitigation
- Two years of persistent disclosure process

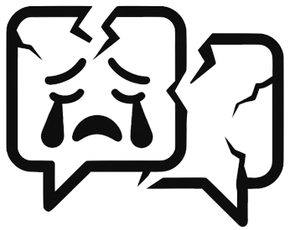
First Comprehensive Analysis

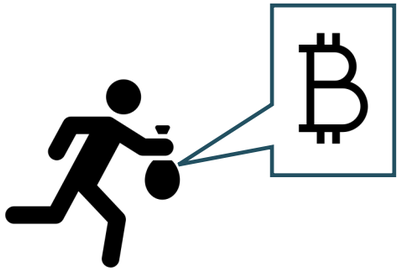
Practical Attacks
& PoCs

Mitigation
& Responsible Disclosure

Our findings

- Breaking integrity on All items (e.g., Profile, Contact List Encrypted DMs…)
- Impersonating to another user
- Breaking confidentiality on Encrypted DMs
- Hijacking micro payment (subset of impersonating)



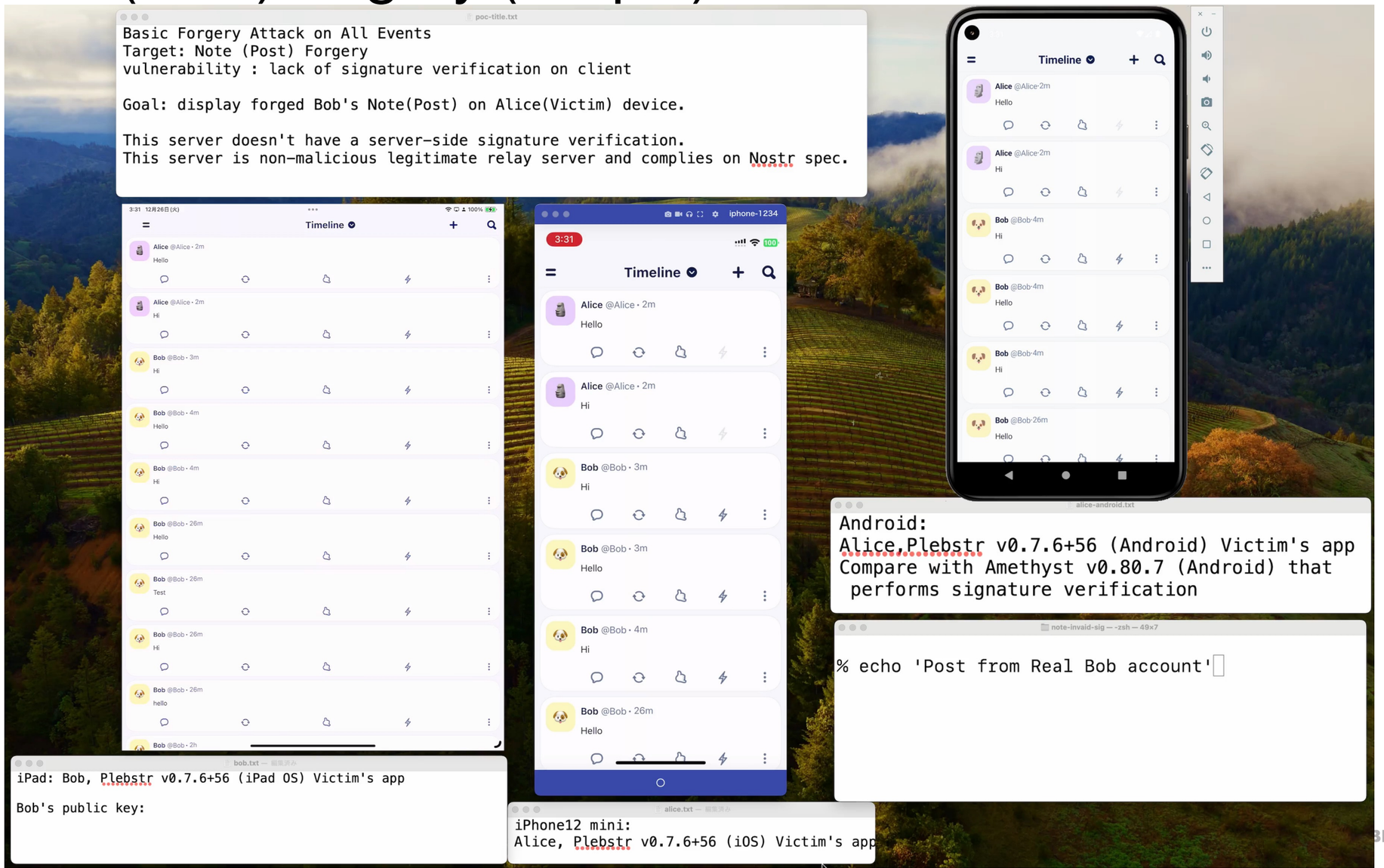


These are not theoretical flaws—they enable practical exploitation

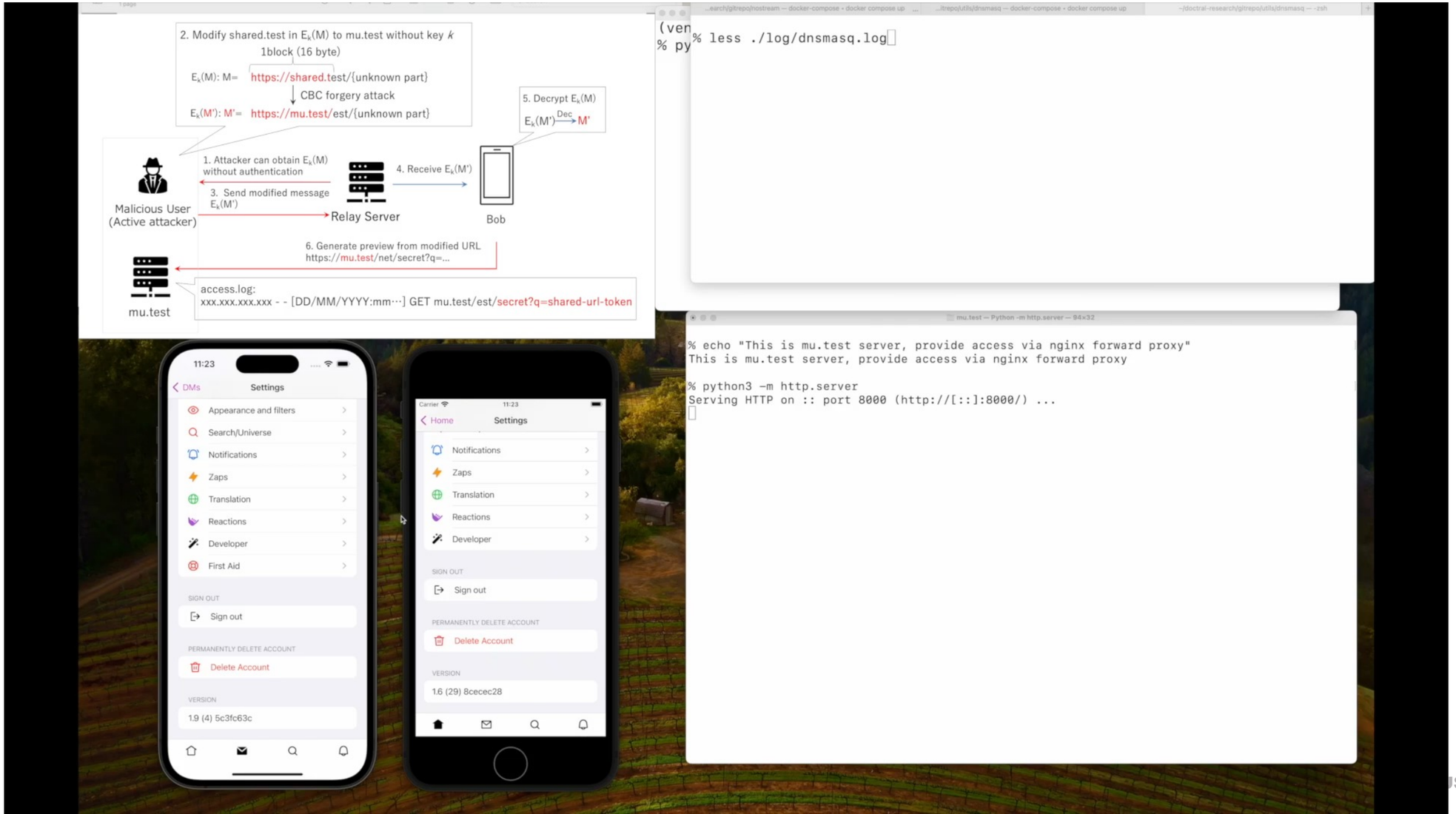
The required threat model varies

Some attacks assume a malicious user; others work under a malicious relay server

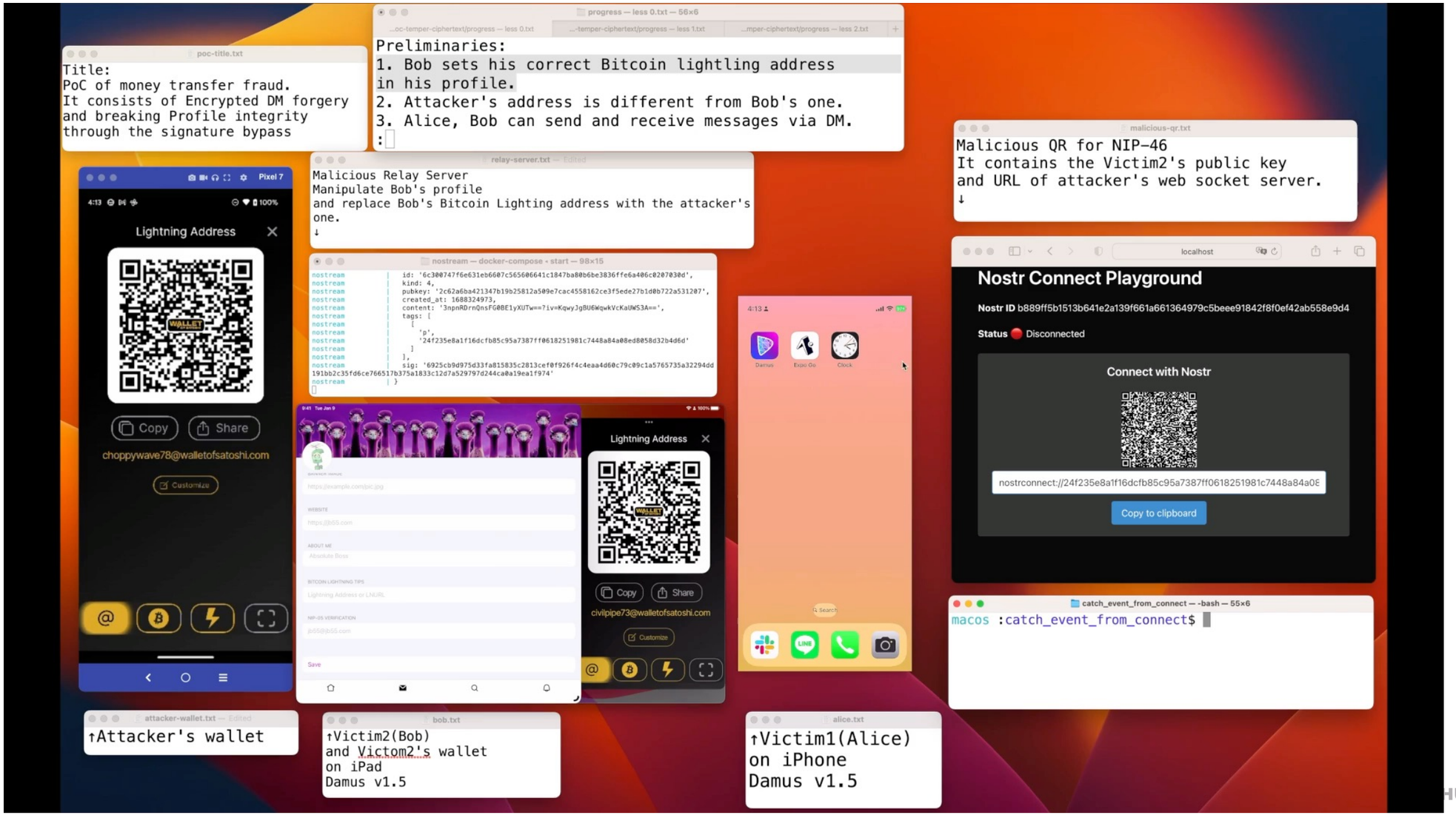
PoC: Note (Post) forgery (simple)



PoC: Encrypted DMs forgery & URL recovery



PoC: Hijacking micro payment ← Profile forgery (cache) & DMs forgery



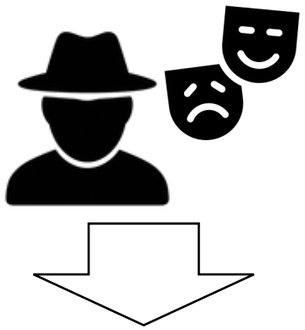
Why does it happen?

Cryptographic protocol design flaw + Implementation flaw

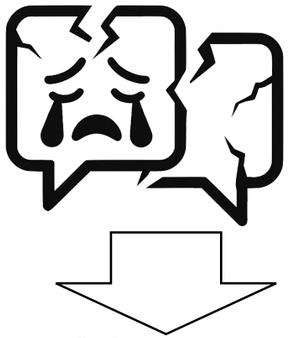
- Breaking integrity on
  All items
- Impersonating to another user

- Breaking confidentiality on
  Encrypted DMs

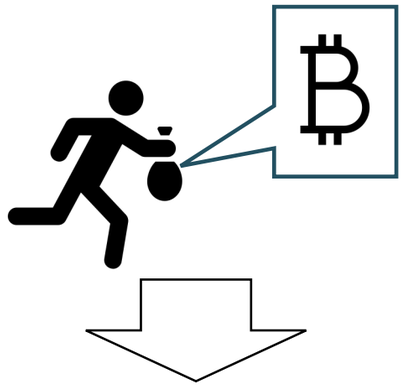
- Hijacking micro payment
  (subset of impersonating)



- Signature verification Bypass



- Lack of key separation
- Receiver-side preview generation



- Verification Bypass

Why does it happen?

Cryptographic protocol design flaw + Implementation flaw

Step by step attack tracing

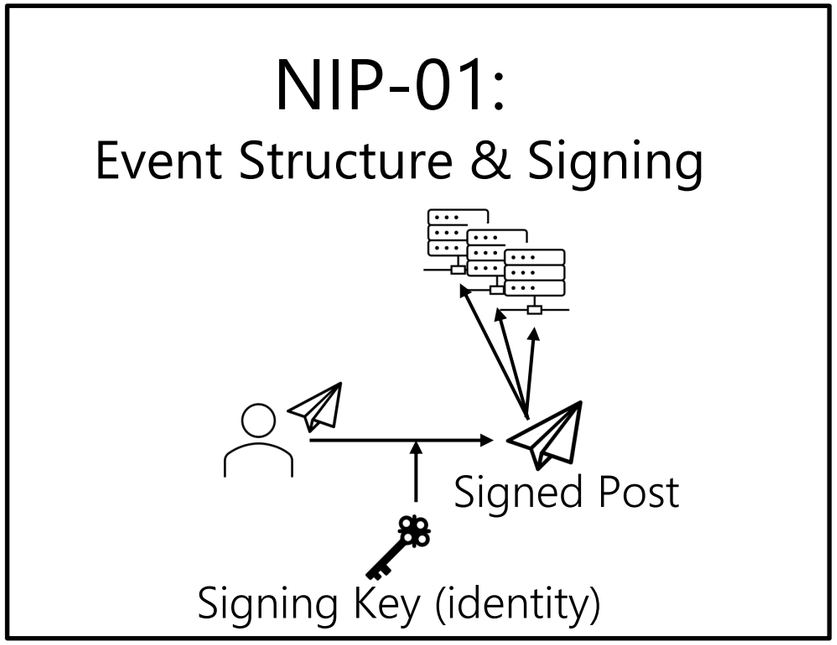
Breaking…

Plaintext integrity
(simple / cache poisoning)

Ciphertext integrity

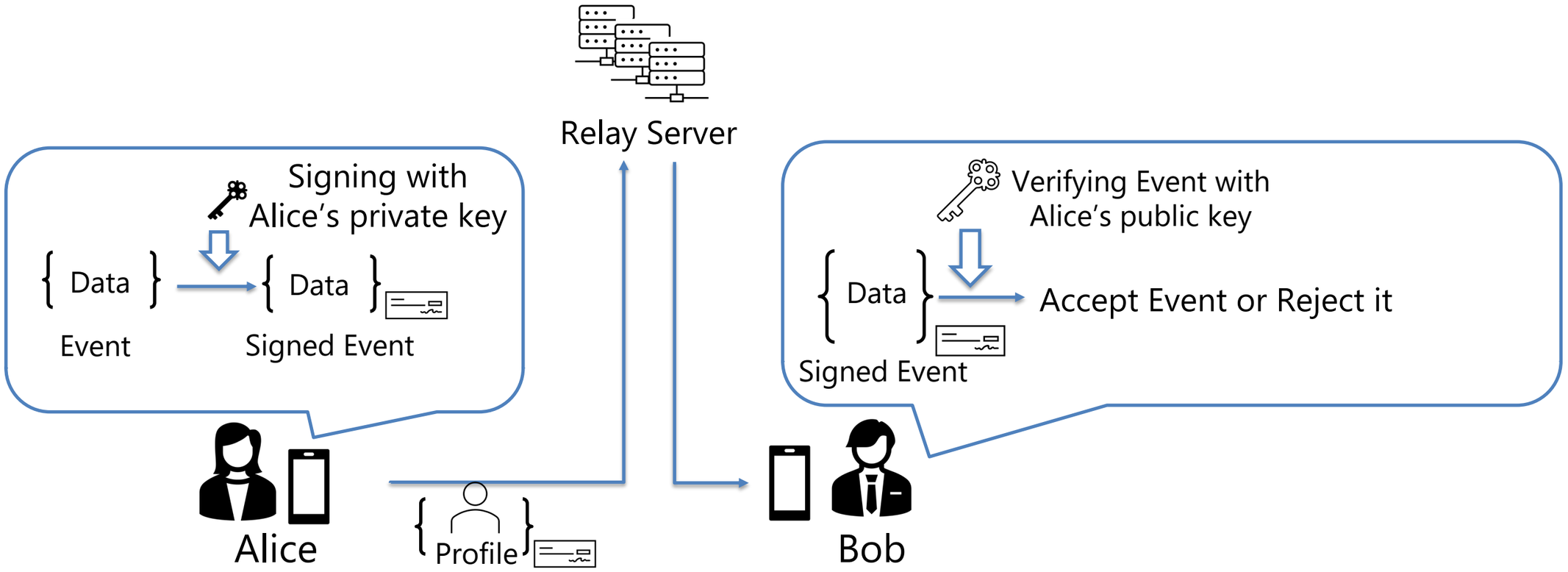
Ciphertext confidentiality

Remark: mandatory signing specification (simplified)



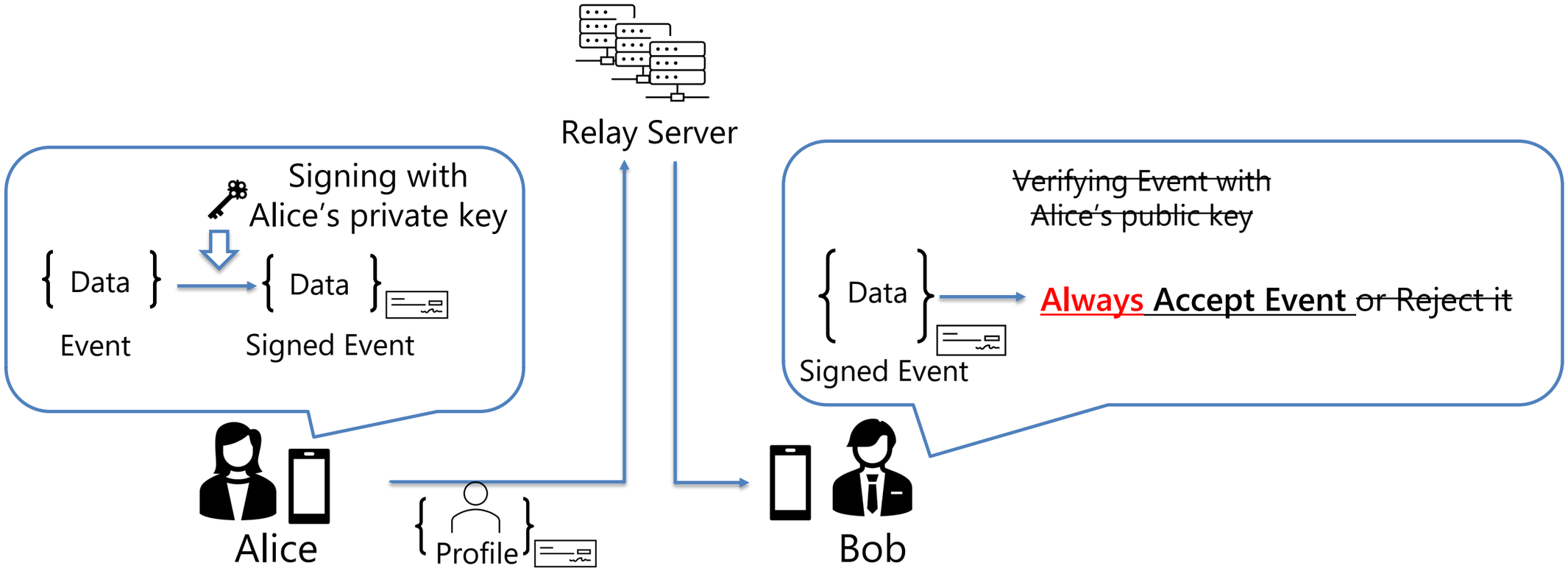
Breaking…
Plaintext integrity
(simple)

Remark: mandatory signing specification (details depending on specification)



Breaking…
Plaintext integrity
(simple)

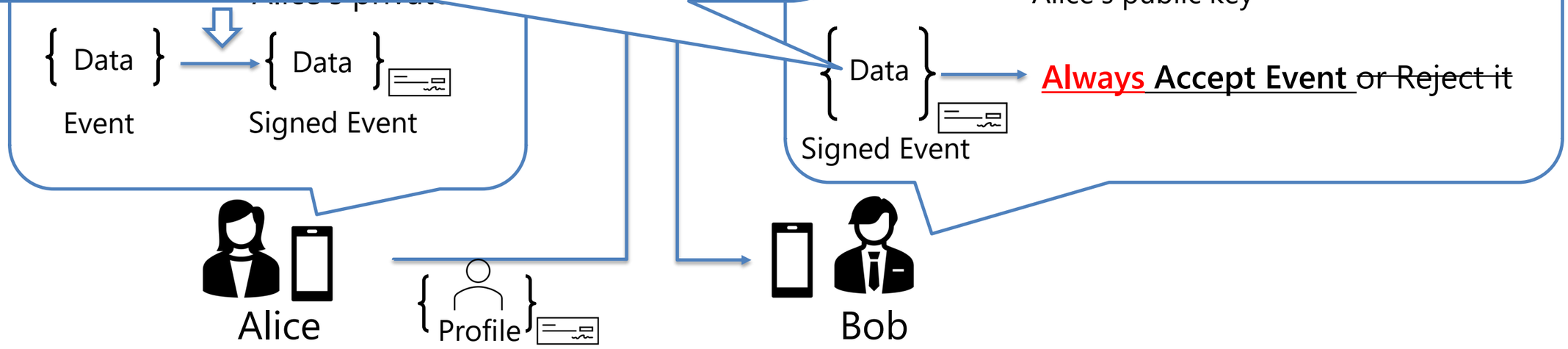
Details depending on many **actual** implementations



y **actual** implementations

Event Type	Data
Profile	Name, Bio, BTC address
Encrypted DM	Encrypted Msg
Post	Plaintext Msg

etc…



Breaking…
Plaintext integrity
(simple)

Details depending on many **actual** implementations

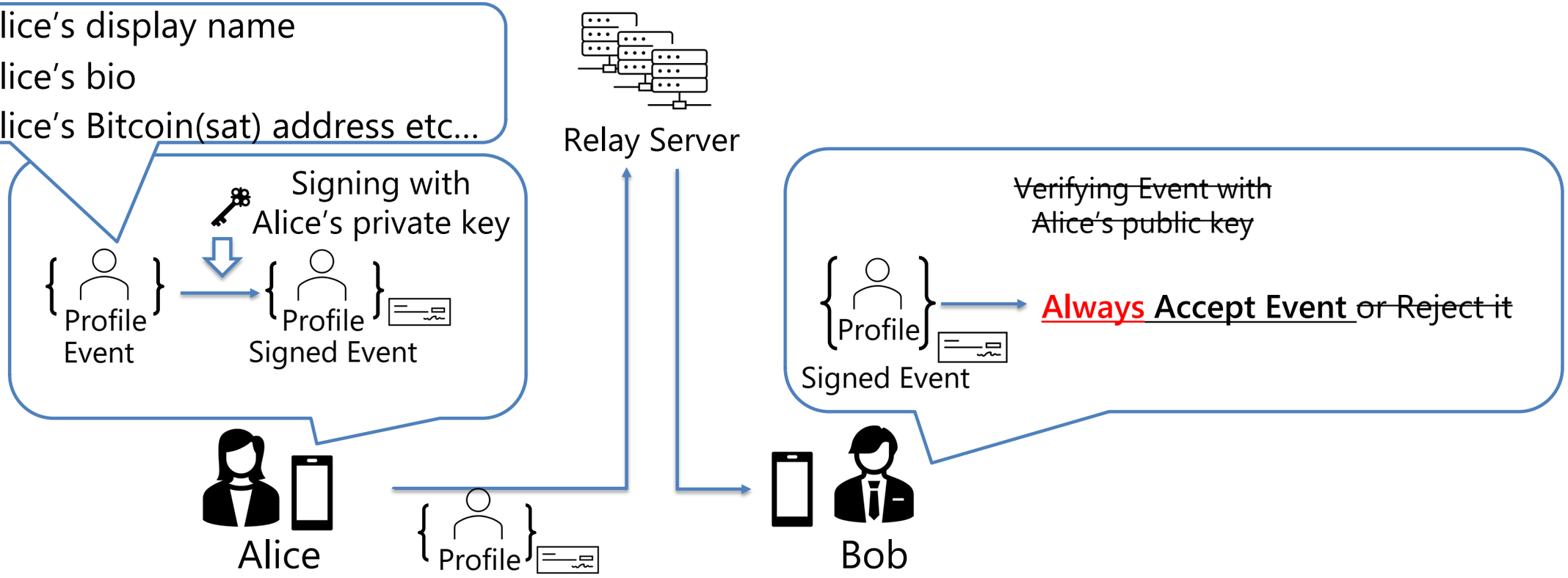
```
541
542     func handle_text_event(sub_id: String, _ ev: NostrEvent) {
543         guard should_show_event(contacts: damus_state.contacts, ev: ev) else {
544             return
545         }
546
547         process_image_metadatas(cache: damus_state.events, ev: ev)
548         damus_state.replies.count_replies(ev)
549         damus_state.events.insert(ev)
550
551         if sub_id == home_subid {
552             insert_home_event(ev)
553         } else if sub_id == notifications_subid {
554             handle_notification(ev: ev)
555         }
556     }
557
```

There is no Verify(Sig) call in the event handling!

Breaking…
Plaintext integrity
(simple)

Case : Alice publishes her Profile & Bob subscribes it

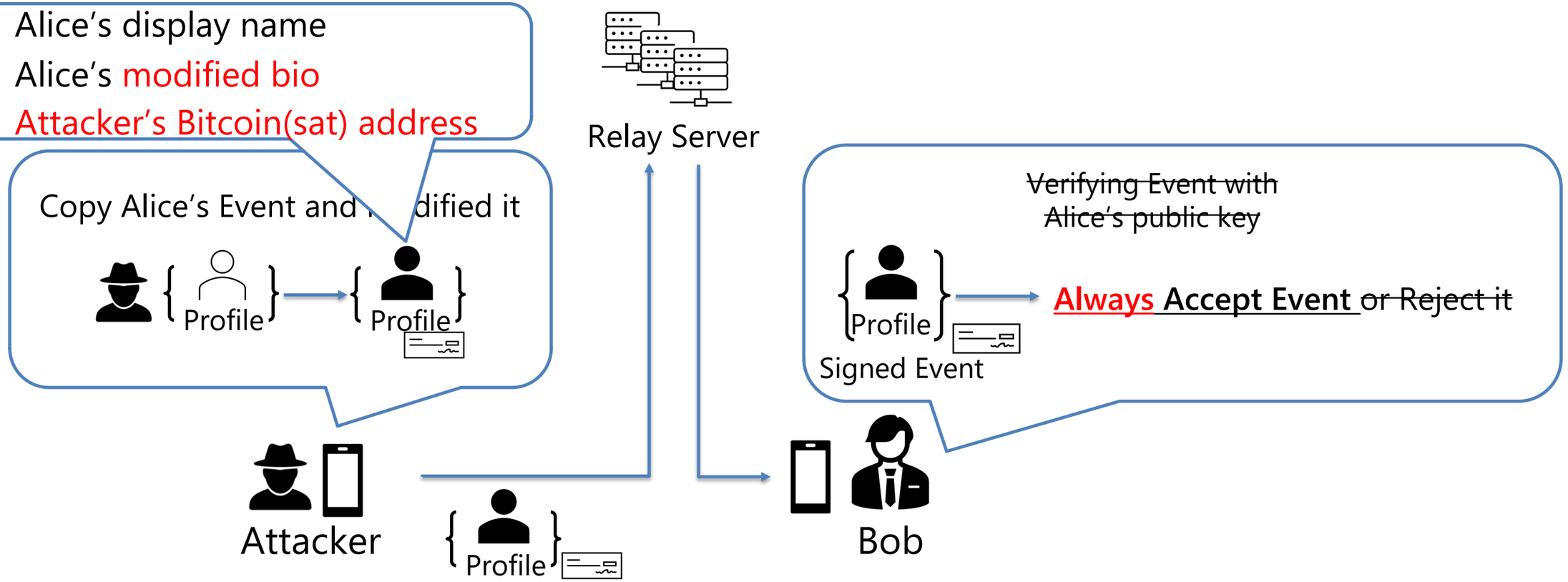
- Alice's display name
- Alice's bio
- Alice's Bitcoin(sat) address etc…



Breaking…
Plaintext integrity
(simple)

Profile Forgery on Plebstr, FreeFrom
Attacker also can publish Alice's Profile

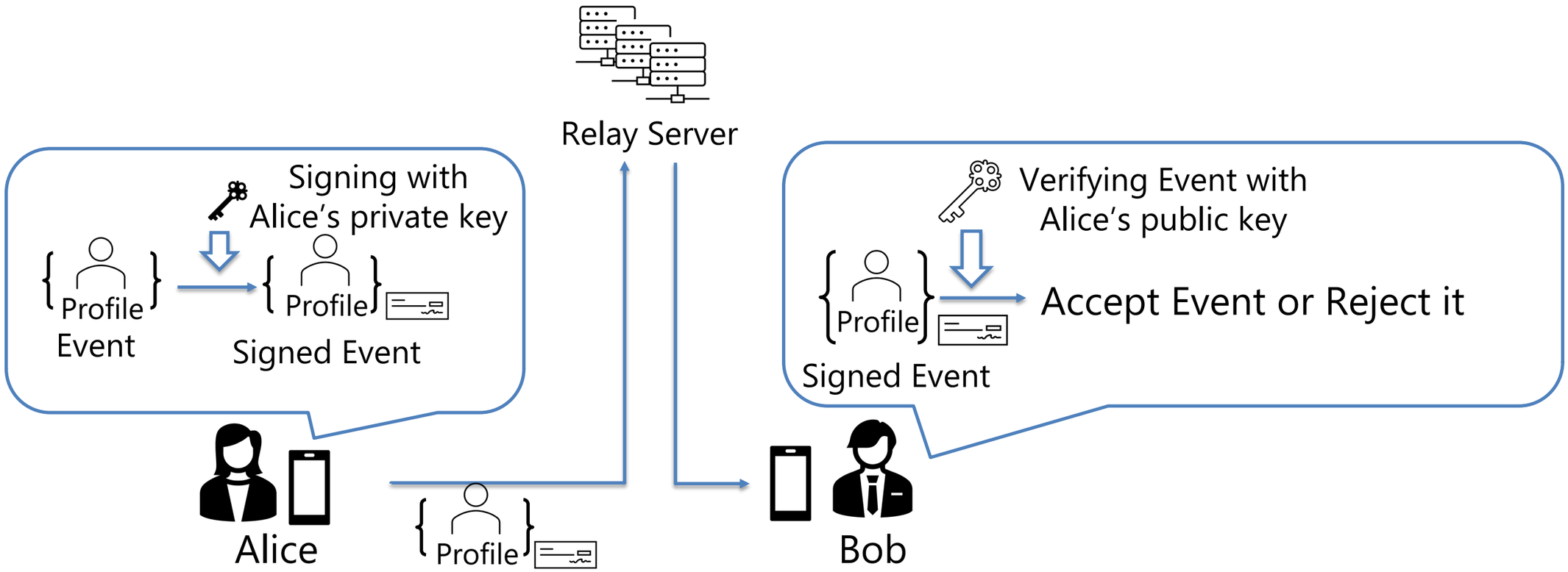
- Alice's display name
- Alice's modified bio
- Attacker's Bitcoin(sat) address



Breaking…
Plaintext integrity
(cache poisoning)

On the Profile validation of Damus (v1.5(8) & v1.6 (29))

Attack on a popular Nostr client



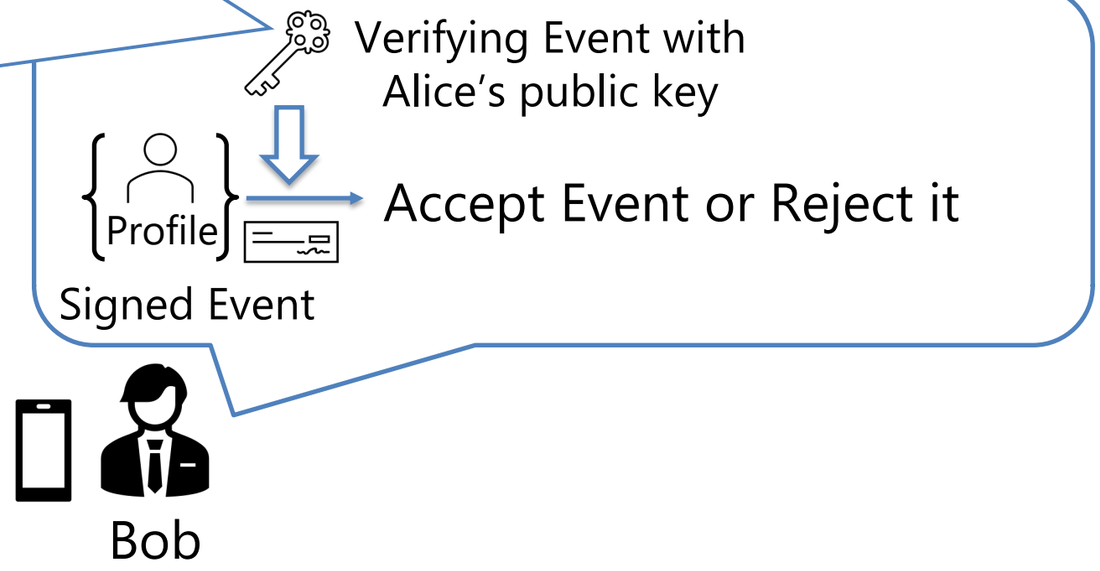
Breaking…
Plaintext integrity
(cache poisoning)

On the Profile validation of Damus (v1.5(8) & v1.6 (29))

Attack on a popular Nostr client

Sig verification in place 😀

```
905  func validate_event(ev: NostrEvent) -> Va
929
930      ok = secp256k1_schnorrsig_verify(ctx,
             &xonly_pubkey) > 0
931      return ok ? .ok : .bad_sig
932  }
```



Breaking…
Plaintext integrity
(cache poisoning)

On the Profile validation of Damus (v1.5(8) & v1.6 (29))

Let‘s see stack trace

Sig verification in place 😀

```
905  func validate_event(ev: NostrEvent) -> Va
929
930      ok = secp256k1_schnorrsig_verify(ctx,
             &xonly_pubkey) > 0
931      return ok ? .ok : .bad_sig
932  }
```

validate_event

↓

Secp256k1.
Schnorr.Verify

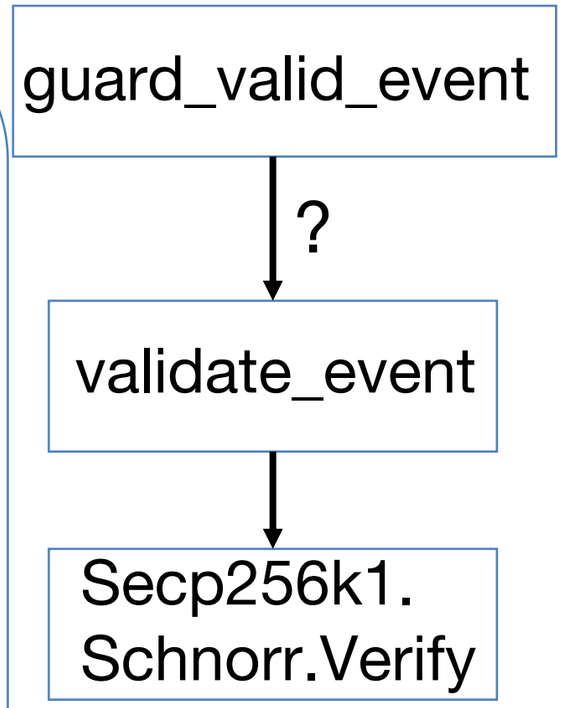
Breaking…
Plaintext integrity
(cache poisoning)

On the Profile validation of Damus (v1.5(8) & v1.6 (29))

Let‘s see stack trace

EventCache & reference …? 😐

```
func guard_valid_event(events: EventCache, ev: NostrEvent, callback: @escaping
    let validated = events.is_event_valid(ev.id)
```



Breaking…
Plaintext integrity
(cache poisoning)

On the Profile validation of Damus (v1.5(8) & v1.6 (29))

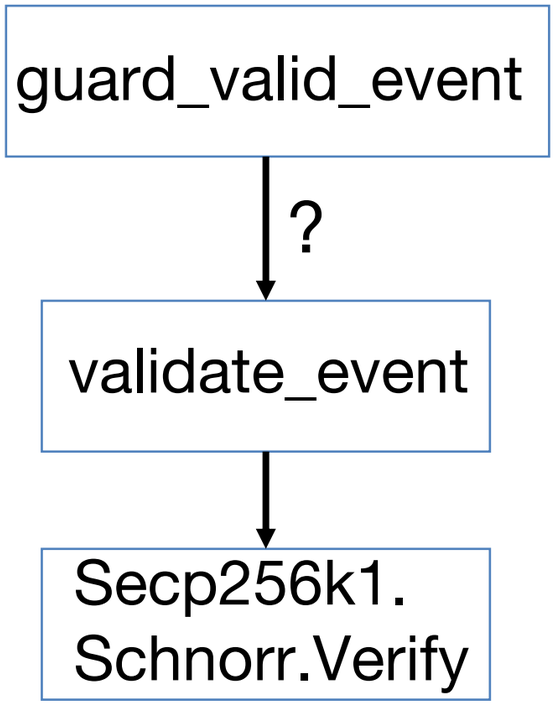
Let‘s see stack trace

EventCache.is_event_valid(ev.id)

Check past signature verification result

- Return **true** if the event is found and the past verification **succeeded**
- Return **false** otherwise.

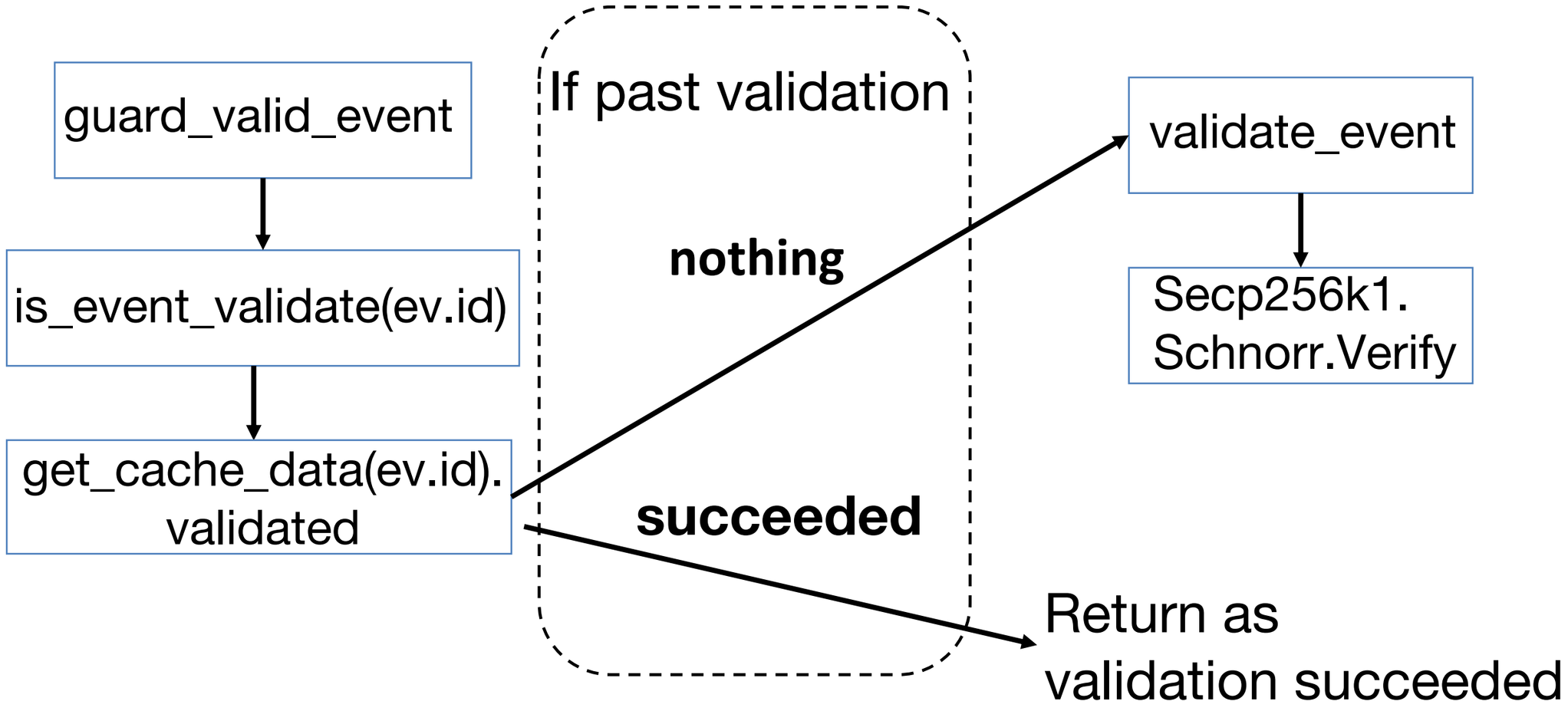
```
157     func get_cache_data(_ evid: String) -> EventData {
158         guard let data = event_data[evid] else {
159             let data = EventData()
160             event_data[evid] = data
161             return data
162         }
163
164         return data
165     }
166
167     func is_event_valid(_ evid: String) -> ValidationResult {
168         return get_cache_data(evid).validated
169     }
```



Breaking…
Plaintext integrity
(cache poisoning)

On the Profile validation of Damus (v1.5(8) & v1.6 (29))

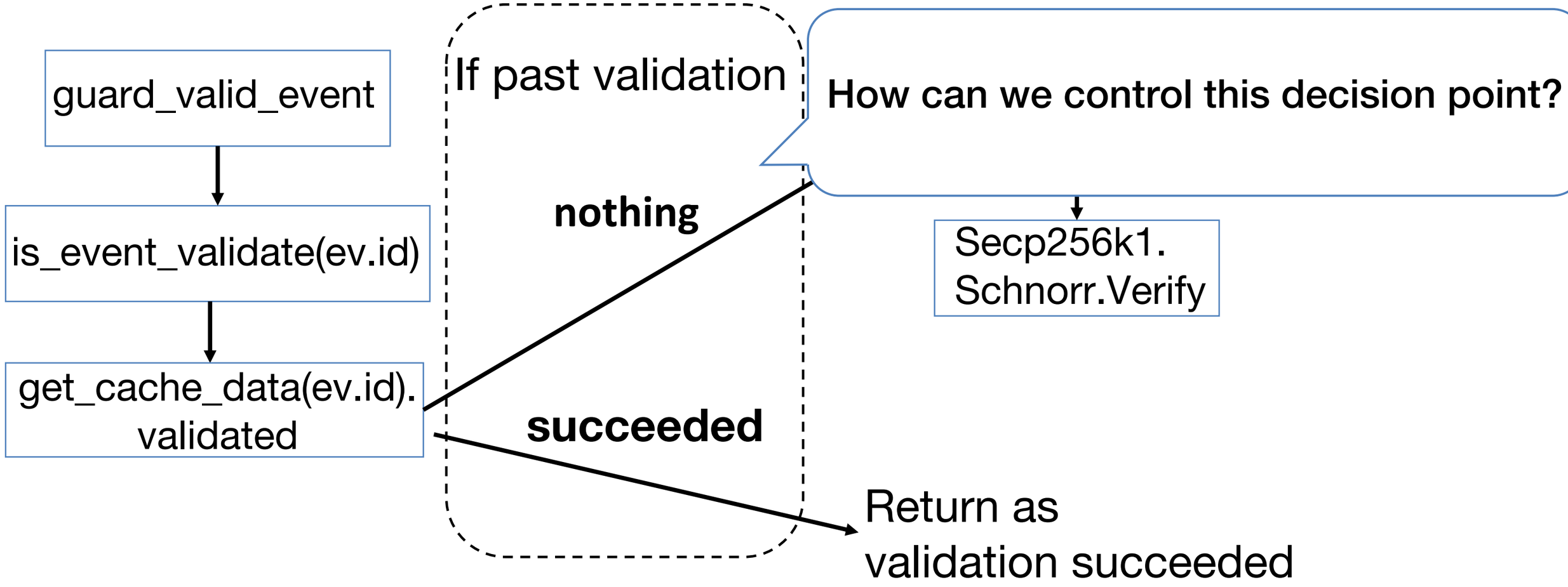
Let‘s see stack trace



Breaking…
Plaintext integrity
(cache poisoning)

On the Profile validation of Damus (v1.5(8) & v1.6 (29))

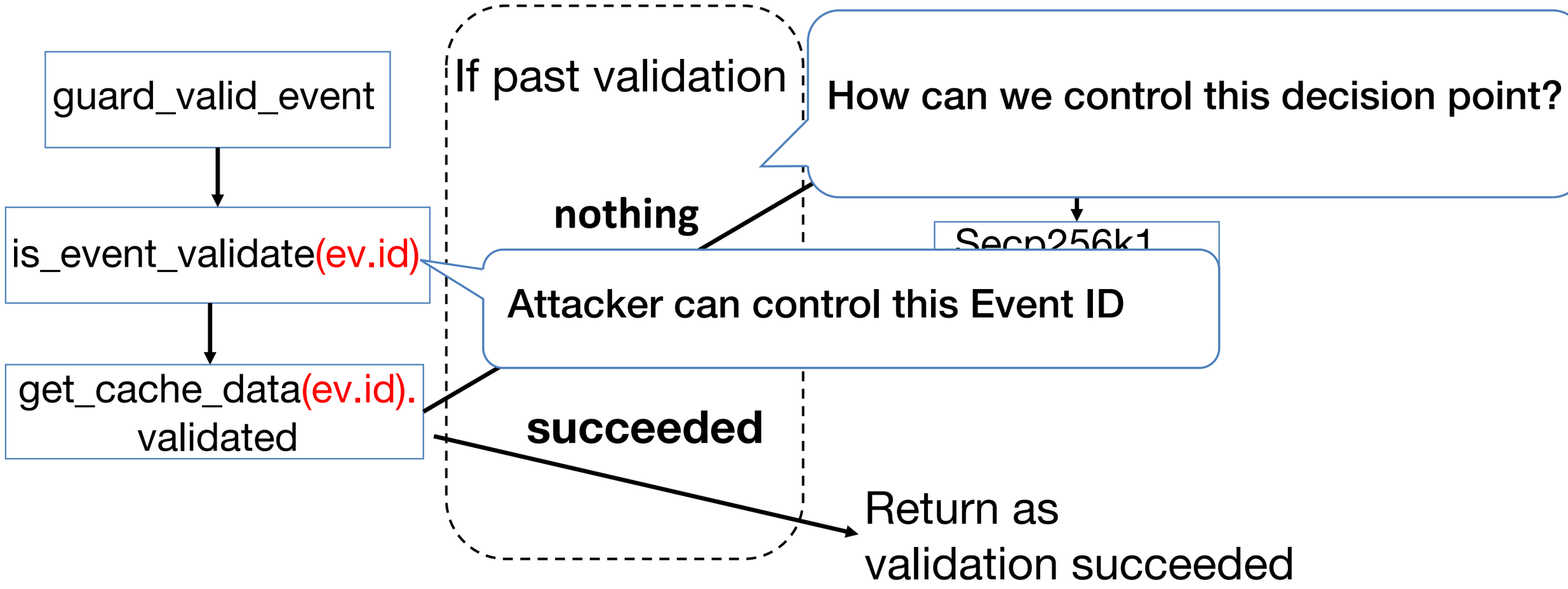
Let‘s see stack trace



Breaking…
Plaintext integrity
(cache poisoning)

On the Profile validation of Damus (v1.5(8) & v1.6 (29))

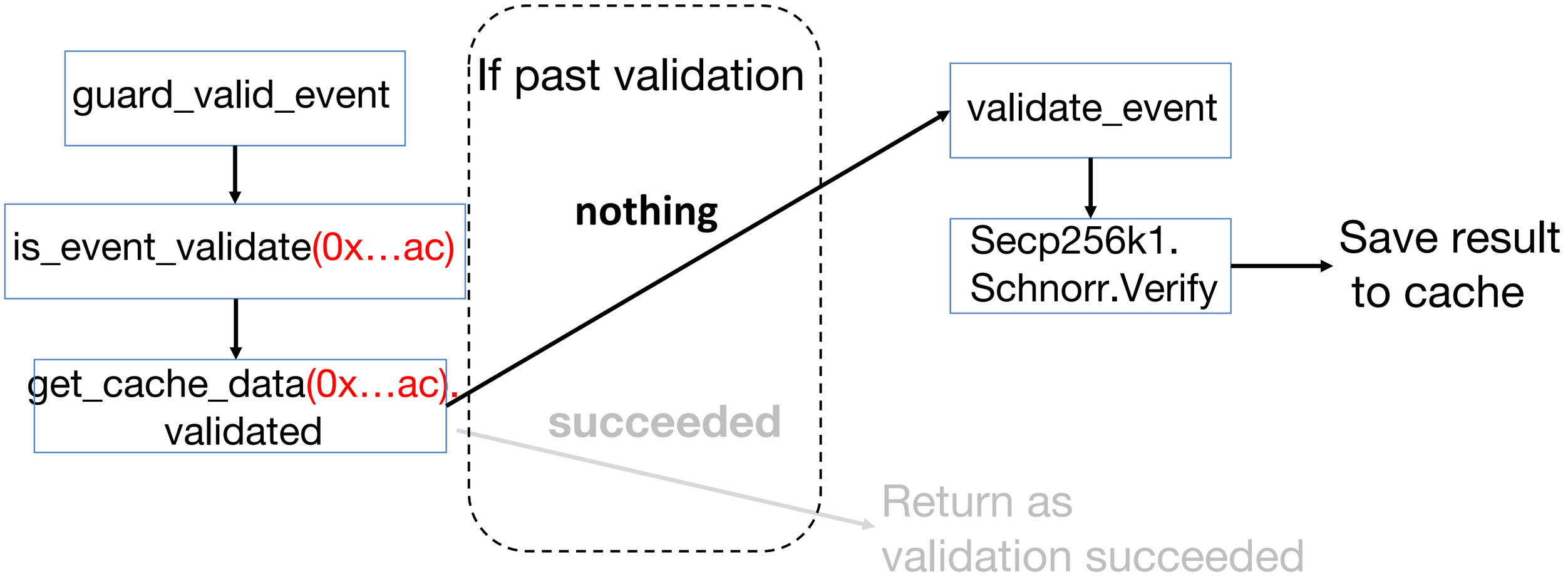
Let‘s see stack trace



Breaking…
Plaintext integrity
(cache poisoning)

On the Profile validation of Damus (v1.5(8) & v1.6 (29))

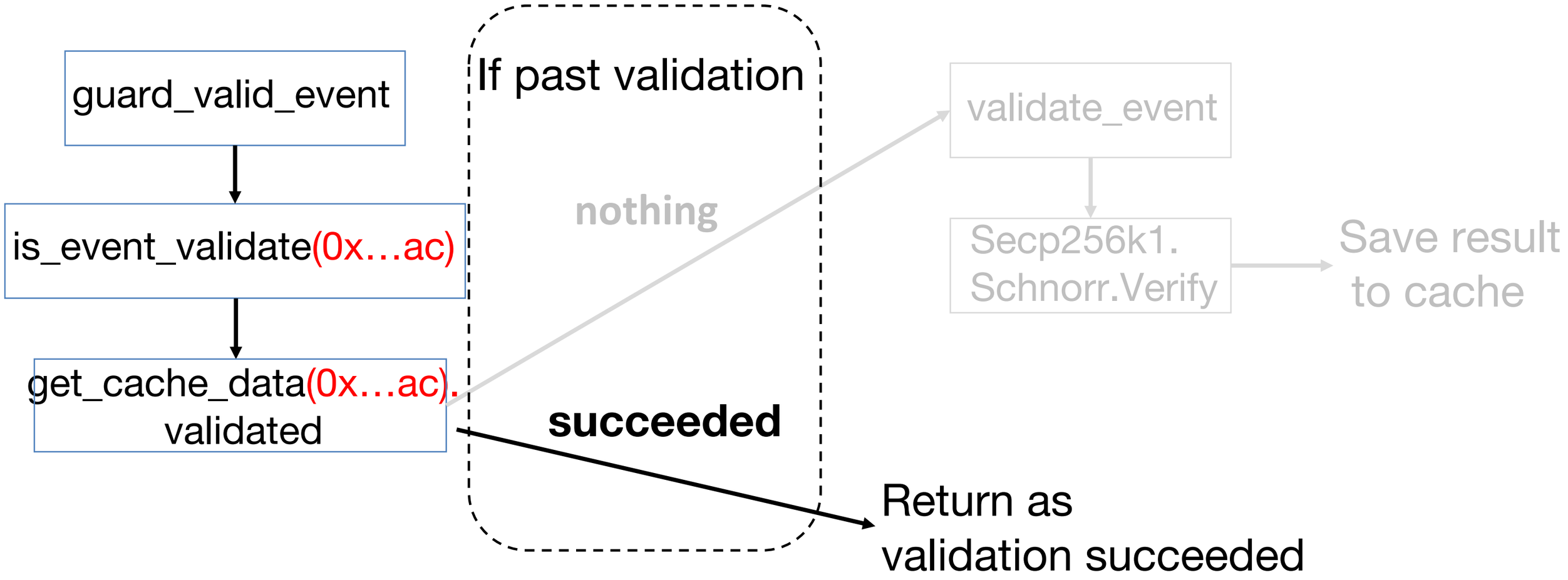
When Bob received an Alice's Event (id== 0x…ac)



Breaking…
Plaintext integrity
(cache poisoning)

On the Profile validation of Damus (v1.5(8) & v1.6 (29))

Attacker sends a fake event with an ID (0x…ac) to Bob

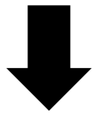


Breaking…
Plaintext integrity
(cache poisoning)

On the Profile validation of Damus (v1.5(8) & v1.6 (29))

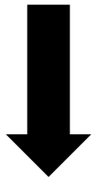
How to derivate Event ID on Nostr

Event ID : ev.id = SHA-256(“0”||{ev.data})



The event ID is a deterministic value derived from ev.data

Root cause: Refer to the cache using the ID without recalculating it



Mitigation

The ID should be recalculated if {ev.data} is modified.

Breaking…
Plaintext integrity
(cache poisoning)

On the Profile validation of Damus (v1.5(8) & v1.6 (29))

Mitigation: Event ID validation

Original: No ID validation

```
750  func guard_valid_event(events: EventCache, ev: NostrEvent,
751      let validated = events.is_event_valid(ev.id)
```

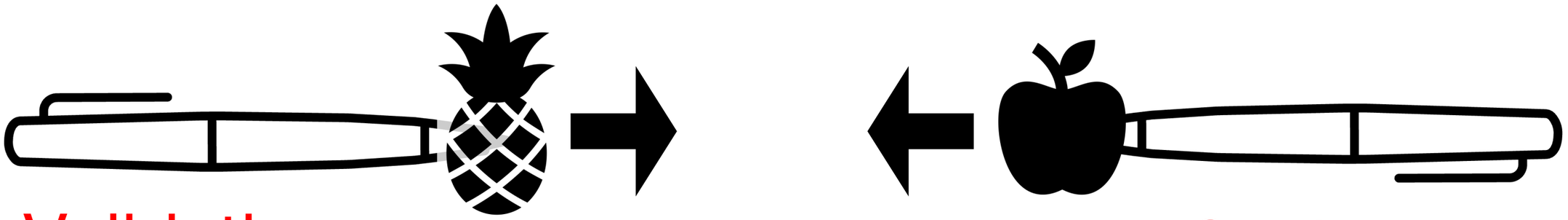
Patched: Ensure ID validation

```
750  func guard_valid_event(events: EventCache, ev: NostrEvent,
751      guard ev.id==calculate_event_id(ev: ev) else {
752          return
753      }
754      let validated = events.is_event_valid(ev.id)
```

Breaking…

Plaintext integrity

Takeaway : Plaintext Integrity

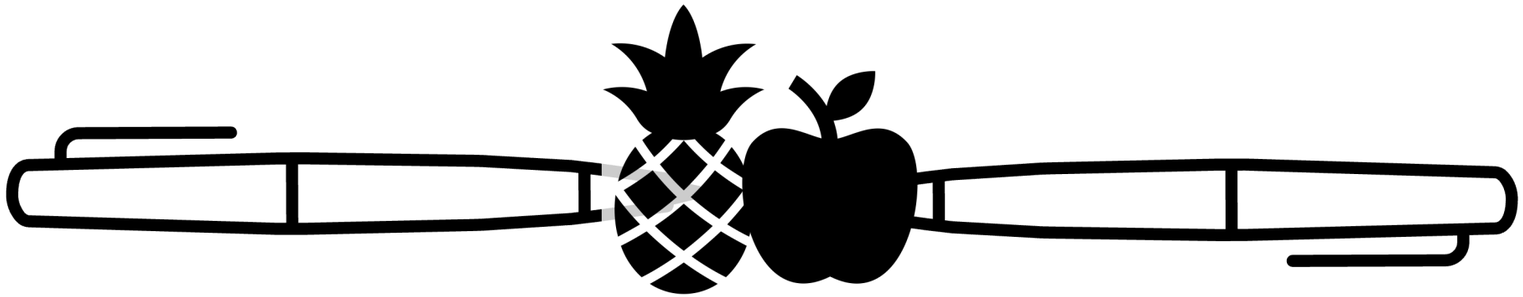


Developer should do integrated security test !

Breaking…

Plaintext integrity

Takeaway : Plaintext Integrity



Authentication Bypass

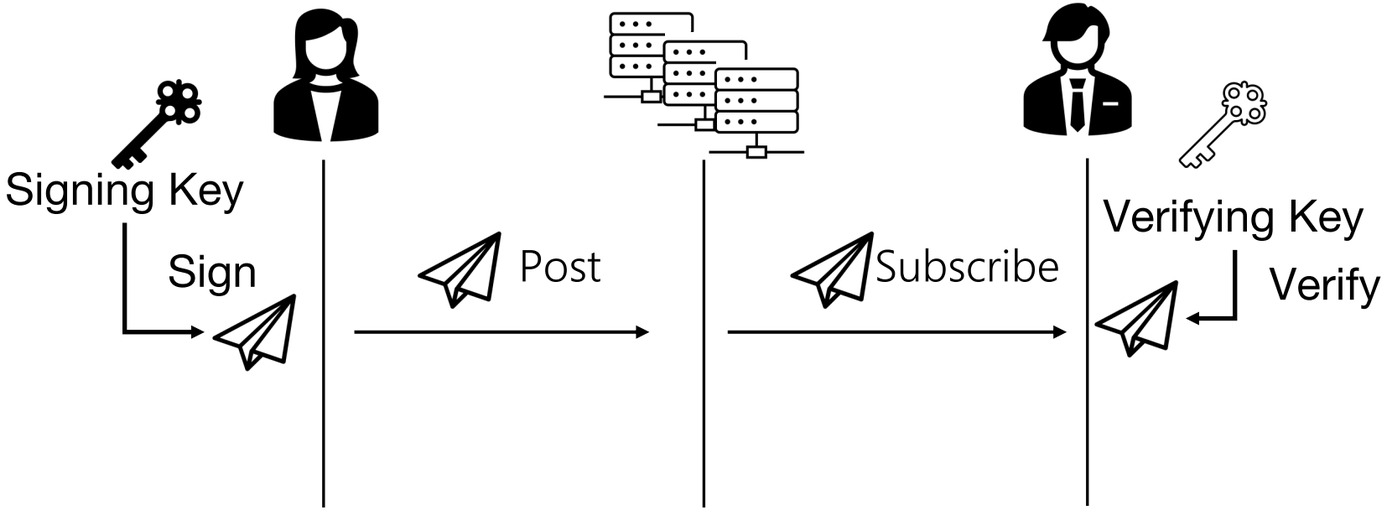
Developer should do integrated security test !

Breaking…

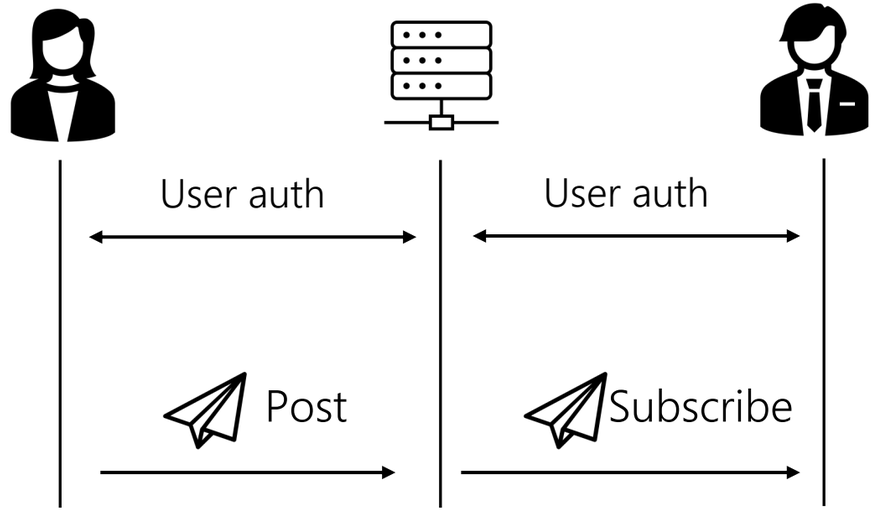
Plaintext integrity

Takeaway : Plaintext Integrity (2)

- In centralized settings, cryptographic flaws often remain “potential risks”
- In self-sovereign decentralized systems like Nostr, they become immediately exploitable
  - Nostr does not have centralized authority
  - Nostr does not provide user authentication by default



Nostr



Centralized SNS

Step by step attack tracing

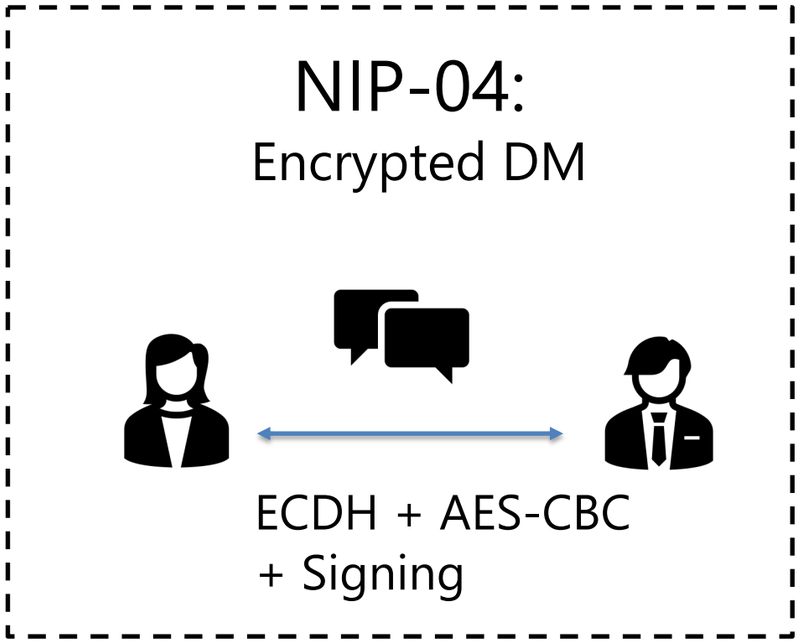
Breaking…

Plaintext integrity
(simple / cache poisoning)

Ciphertext integrity

Ciphertext confidentiality

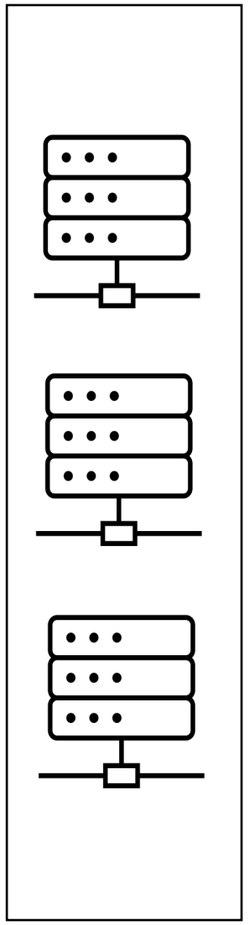
Remark: Encrypted Direct Messages specification (simplified)



Encrypted DM Spec

Alice's Public key (Verifying key)

Alice's Private key (Signing key)



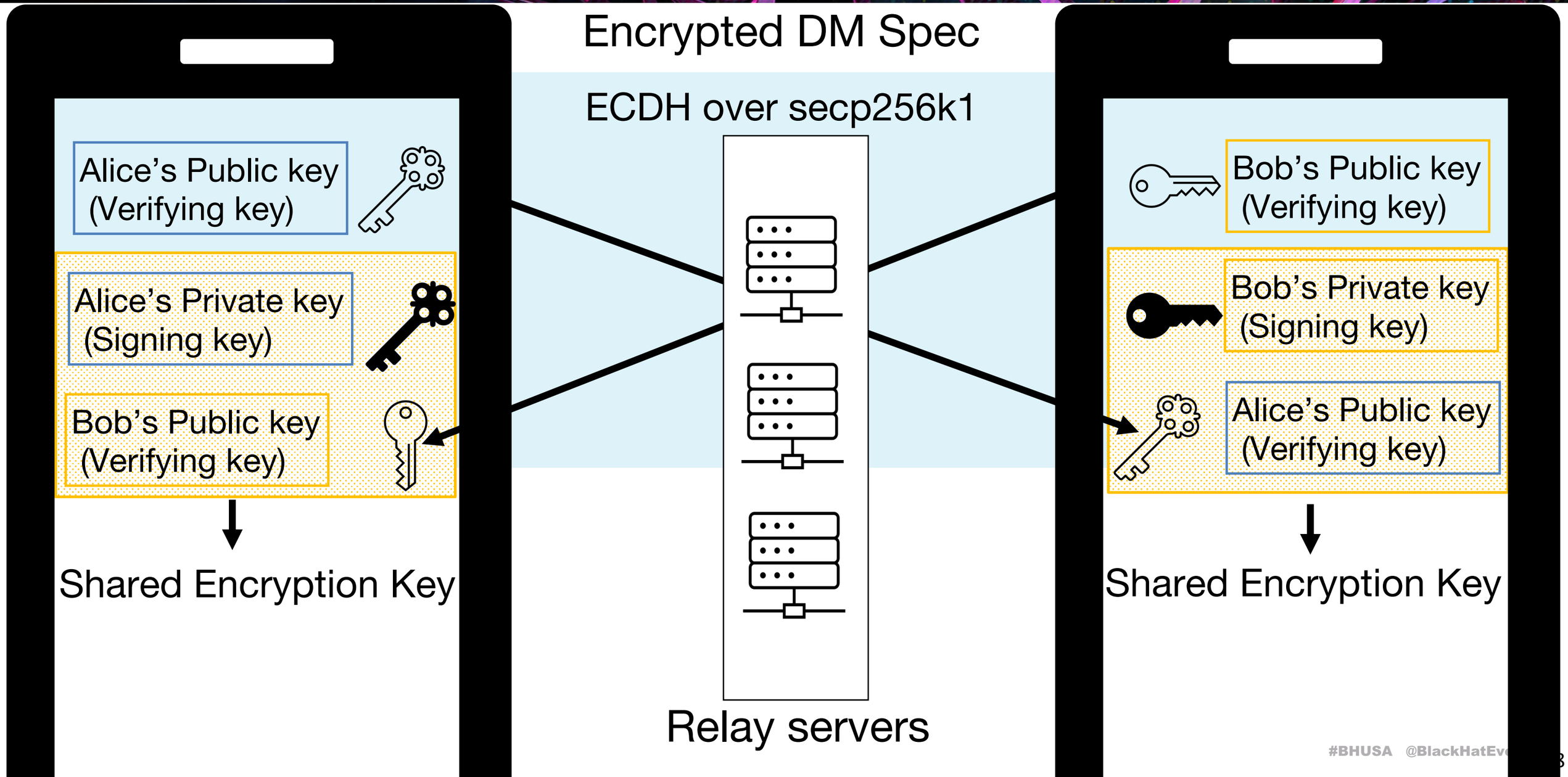
Relay servers

Bob's Public key (Verifying key)

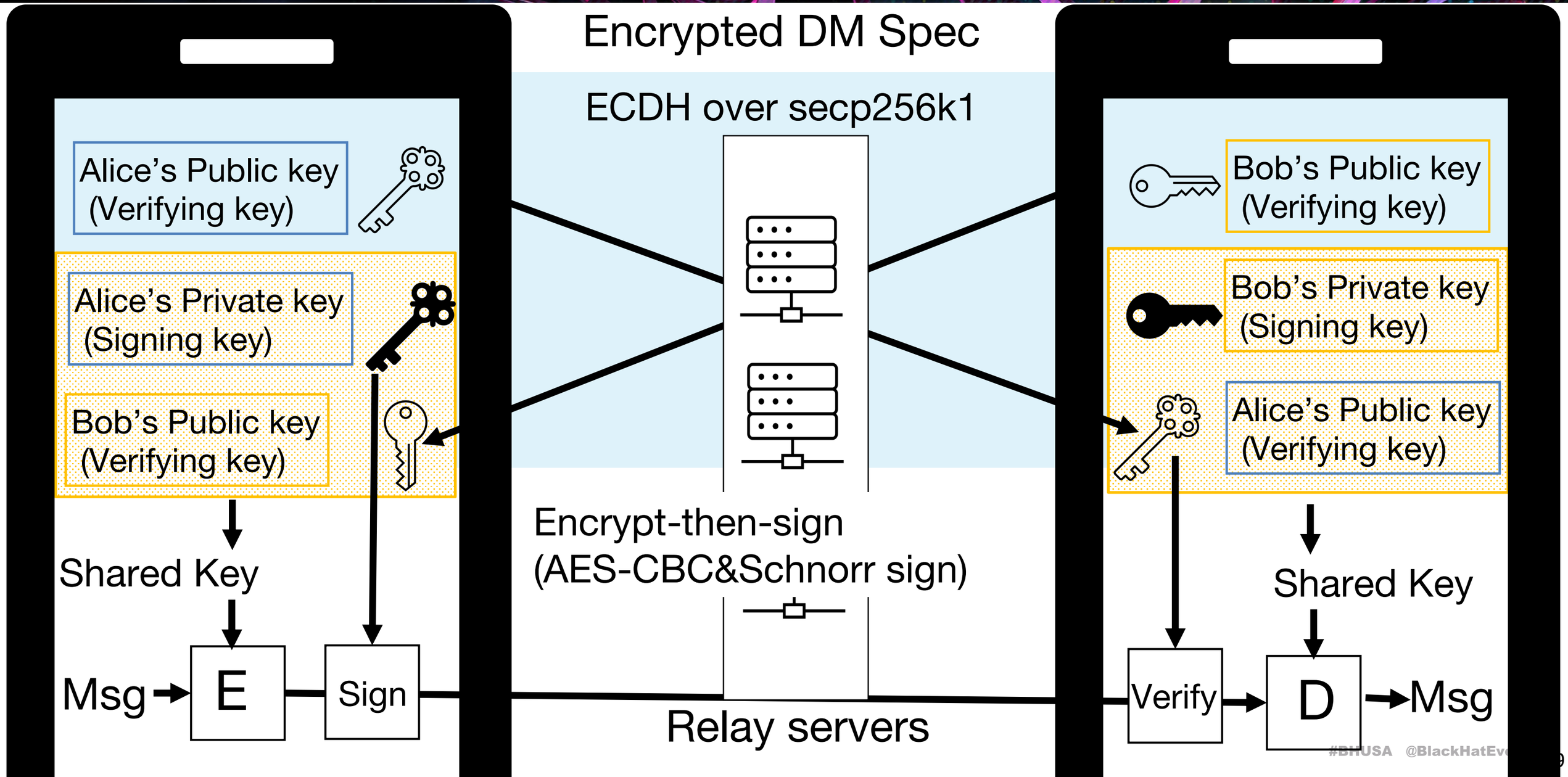
Bob's Private key (Signing key)

Encrypted DM Spec

ECDH over secp256k1

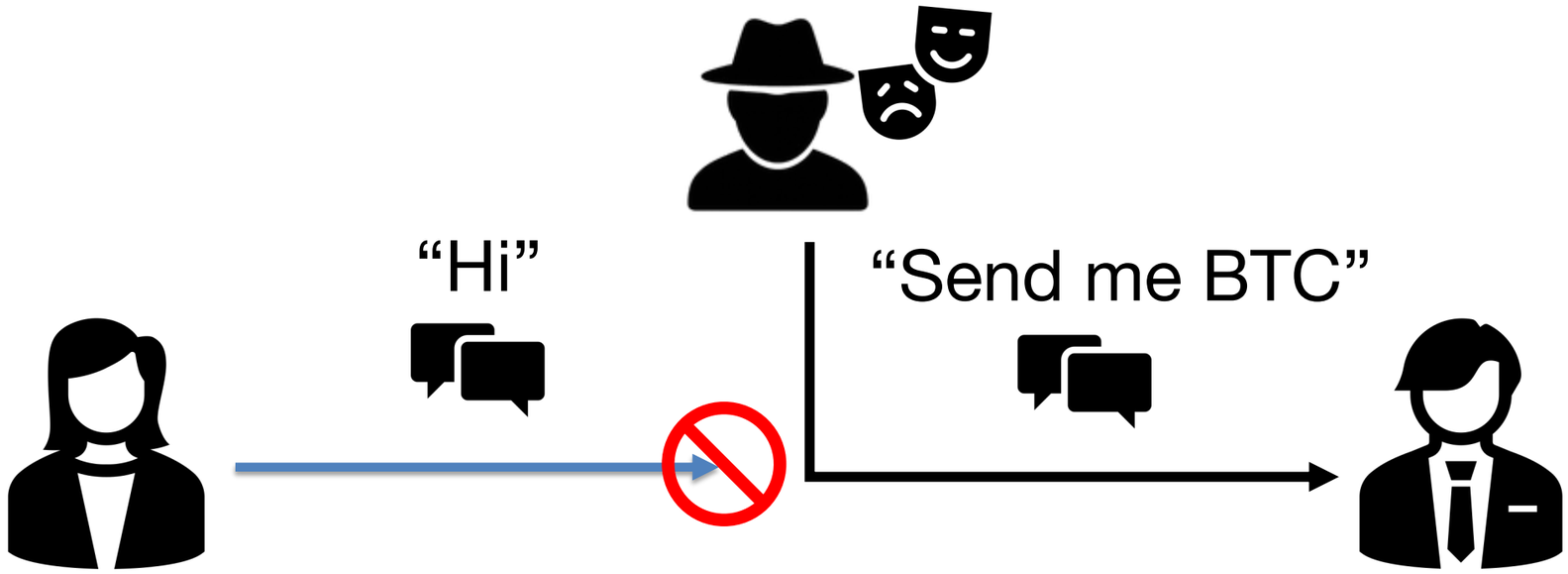


Encrypted DM Spec



Encrypted DM Forgery

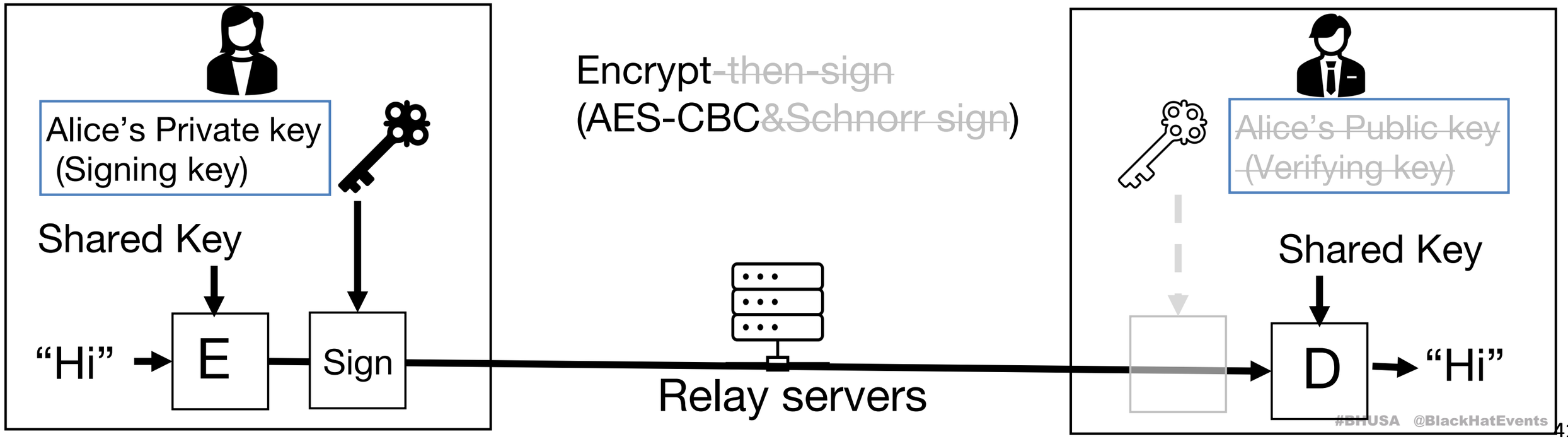
Attacker's Goal : Change decrypted Msg to attacker's Msg_{adv}
e.g., "Send me BTC"



ECDH + AES-CBC
+ Signing

Encrypted DM Forgery

Assumption1: Signature verification is skipped **on the implementation**
(explained earlier)

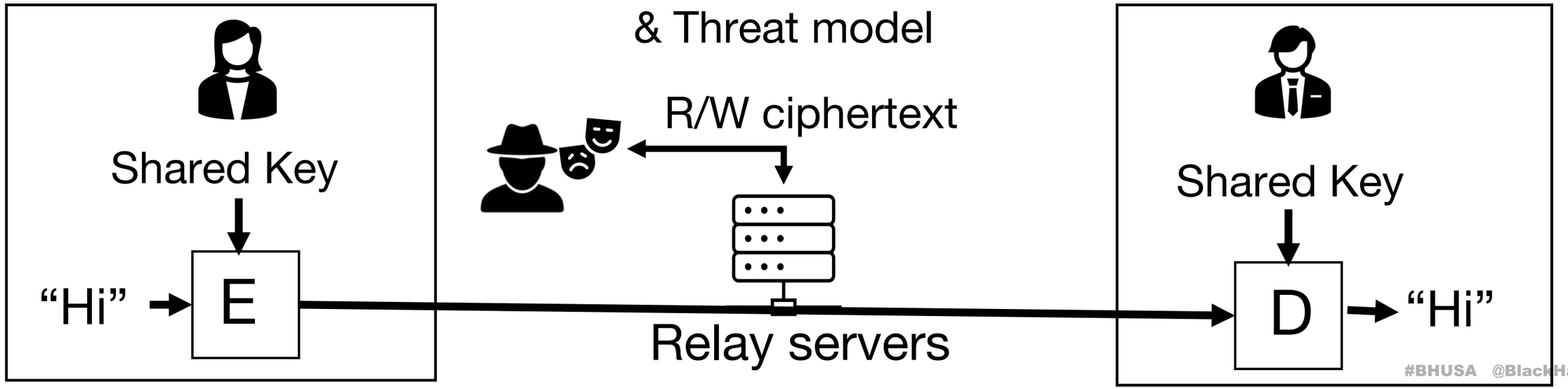


Encrypted DM Forgery

Assumption2: Threat model

- Attacker is a user of Nostr
- Attacker cannot read/write to “Shared Key”
- Attacker can **freely fetch ciphertext** from relay relays
  Nostr does not include user authentication on servers by default

Simplified encryption specs
(AES-CBC)
& Threat model

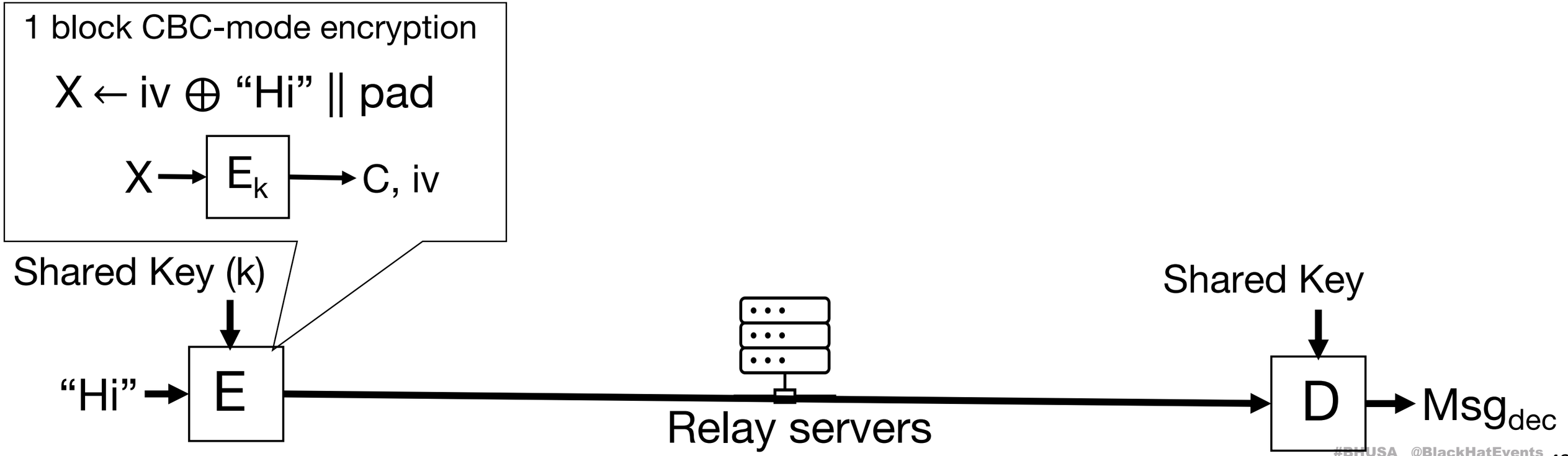


Encrypted DM Forgery

Problem : Verification bypass is not enough to achieve practical forgery on DMs

Reason : CBC Allows Bit Flipping – But decryption result blinds for the attacker

(simplified) Bit flipping on Message Encryption

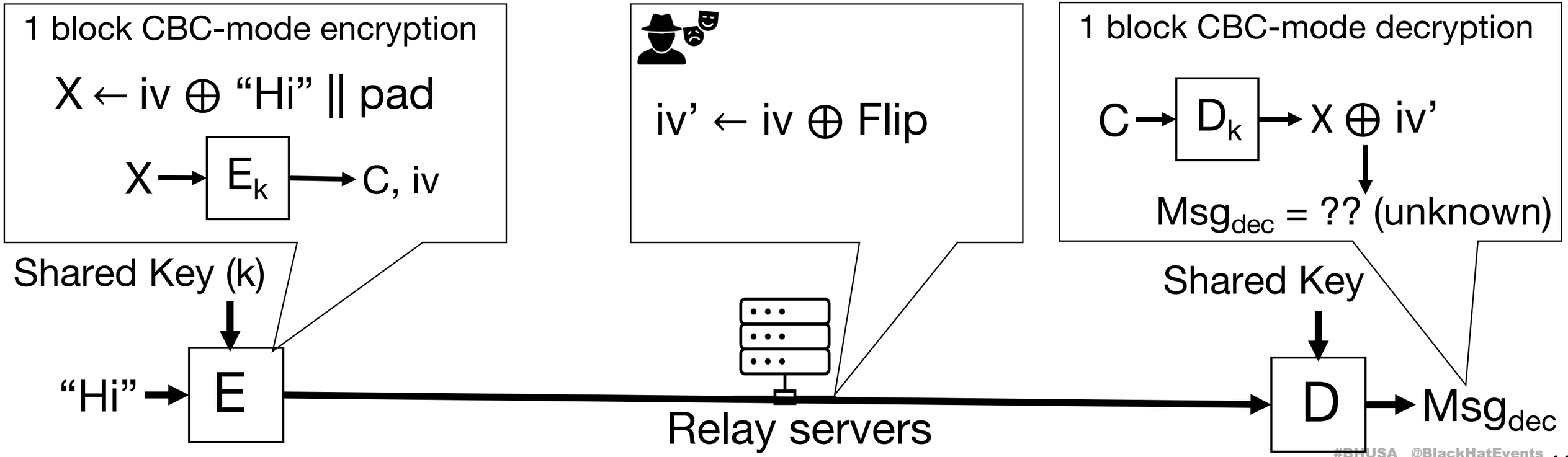


Encrypted DM Forgery

Problem : Verification bypass is not enough to achieve practical forgery on DMs

Reason : CBC Allows Bit Flipping – But decryption result blinds for the attacker

(simplified) Bit flipping on Message Encryption



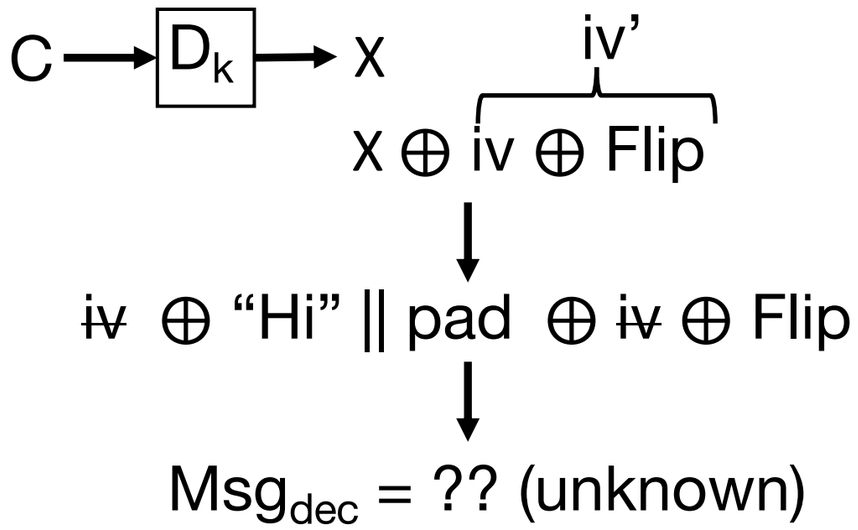
What does the attacker need to control the decryption result?

To craft a forged ciphertext, the attacker needs a reference point:

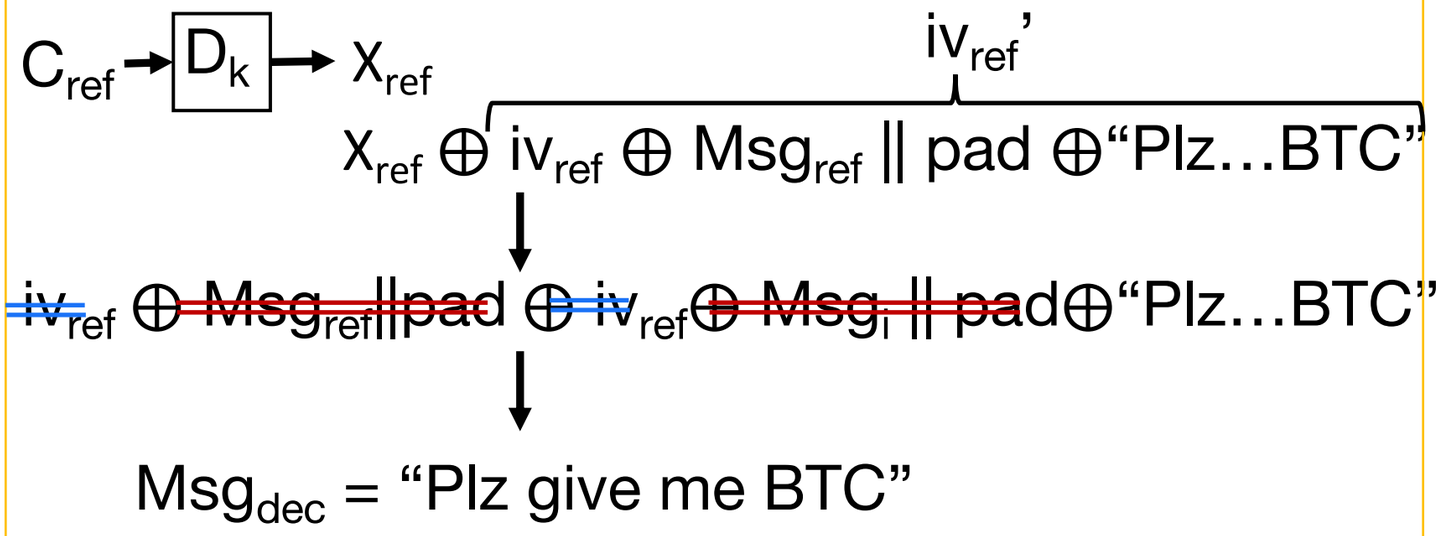
→ a known plaintext/ciphertext (C_{ref}, Msg_{ref}) pair <u>with the same shared key (k)</u>

Cf. Encryption: $X \leftarrow iv \oplus Msg \,||\, pad \quad C \leftarrow E_k(X)$, send iv & C

Random bit-flipping forgery



Practical forgery using a known (C_{ref}, Msg_{ref}) pair



Move from Bit Flipping Forgery to Controlled Practical Forgery

Random bit-flipping forgery

- No decryption knowledge
- Can’t control decrypted message
- Just makes noise

Practical forgery using a known (C_{ref}, Msg_{ref}) pair

- Known plaintext/ciphertext block
- XOR trick enables precision
- Delivers chosen message to victim

Move from Bit Flipping Forgery to Controlled Practical Forgery

Random bit-flipping forgery

- No decryption knowledge
- Can't control decrypted m
- Just makes noise

Problem:
How can we get it ?

Practical forgery using a known (C_{ref}, Msg_{ref}) pair

- Known plaintext/ciphertext block
- XOR trick enables precision
- Delivers chosen message to victim

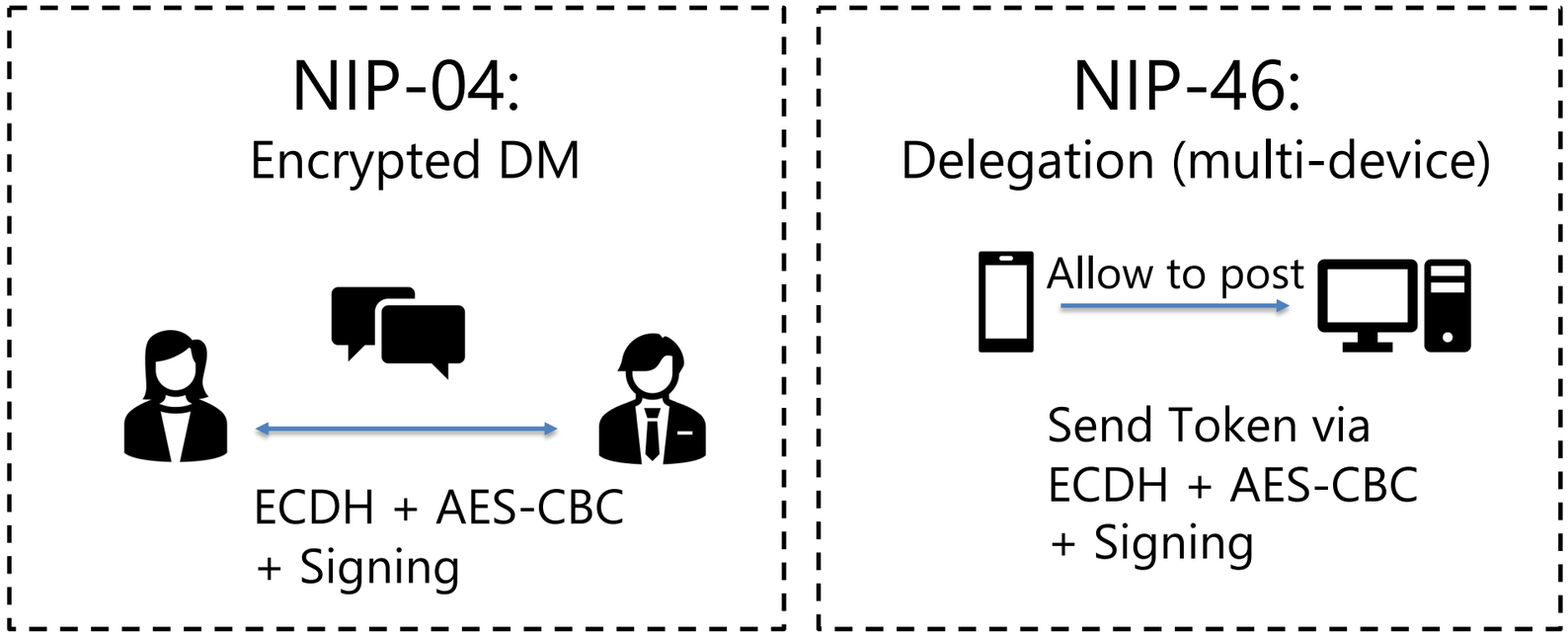
Encrypted DM Forgery via Cross Protocol Attack

Solution : Breaking the Barrier via “Cross Protocol” Attack

Observation:

Delegation (NIP-46) uses same keying & encryption algorithms as DMs (NIP-04)

NIP-46 encrypts **known metadata** using the ***same shared key*** as DMs (NIP-04)



Encrypted DM Forgery via Cross Protocol Attack

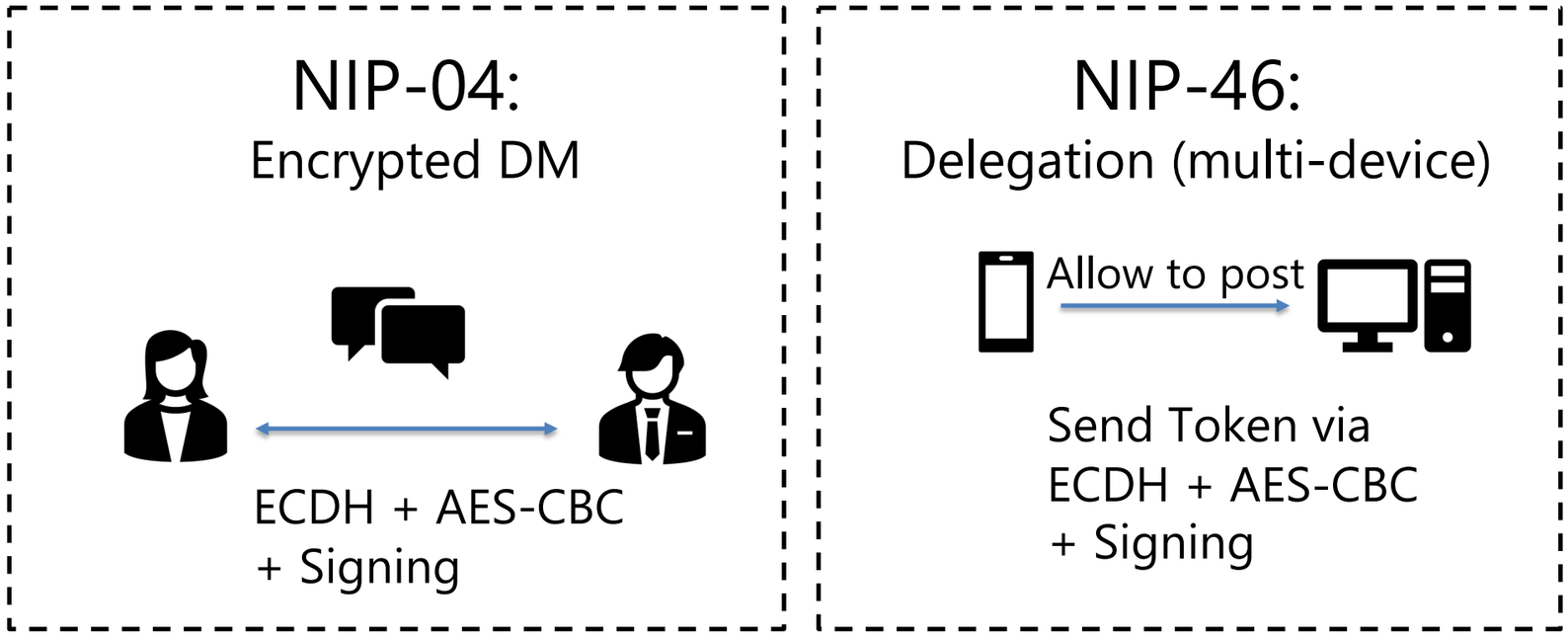
Solution : Breaking the Barrier via “Cross Protocol” Attack

Observation:

Delegation (NIP-46) uses same keying & encryption algorithms as DMs (NIP-04)

NIP-46 encrypts **<u>known metadata</u>** using the ***<u>same shared key</u>*** as DMs (NIP-04)

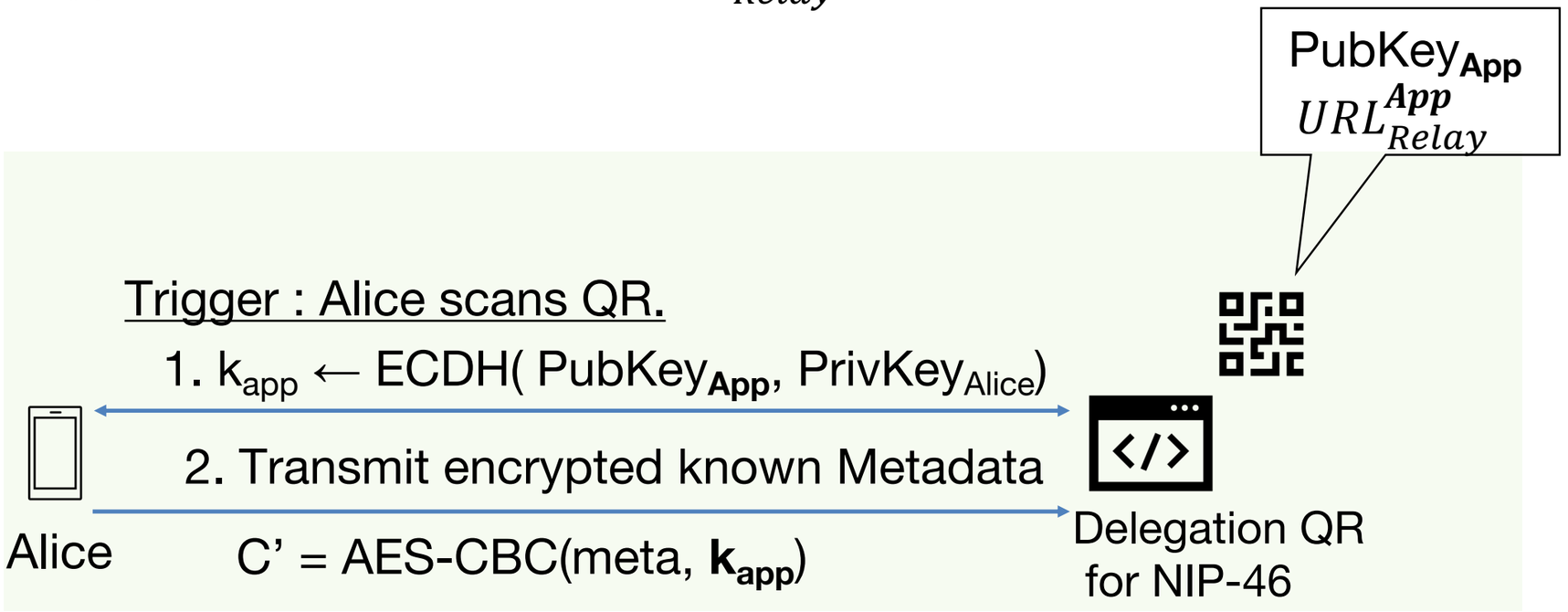
→ makes known plaintext → makes known ciphertext



Encrypted DM Forgery via Cross Protocol Attack

Normal Delegation initial sequence

- ECDH with the public key obtained from the QR
- Sends encrypted known metadata to URL_{Relay}^{App} from the QR



Encrypted DM Forgery via Cross Protocol Attack

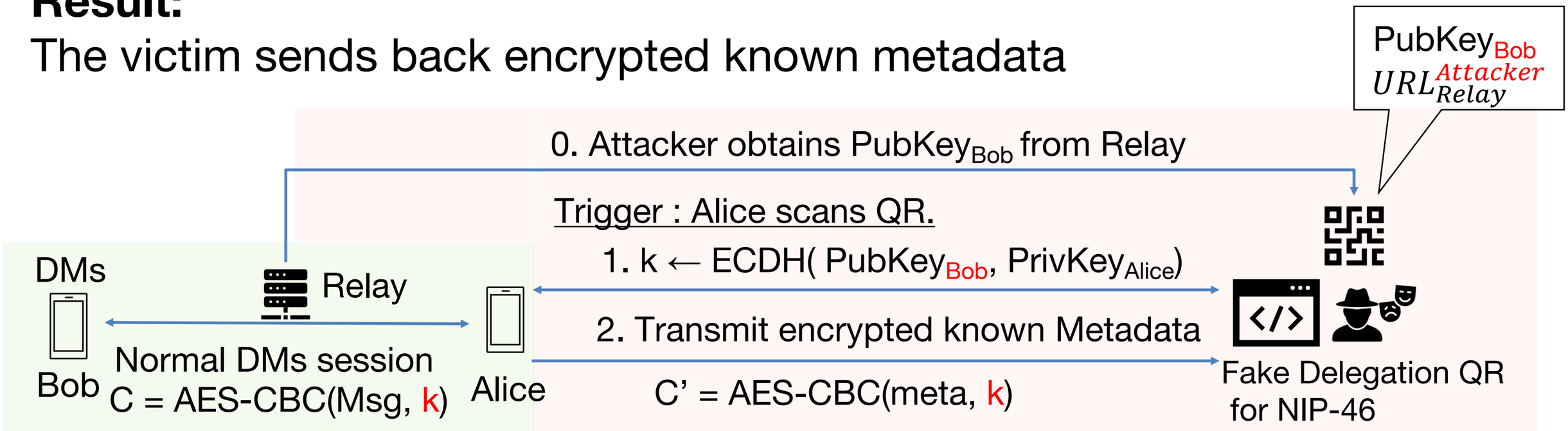
Strategy:

The attacker starts a NIP-46 session with the victim (as a fake delegation app)

The attack puts $PubKey_{Bob}$ to the QR

Result:

The victim sends back encrypted known metadata



Takeaway : Ciphertext Integrity

- **Should use Authenticated Encryption (AE)**
  - E.g., AES-GCM, ChaCha20-Poly1305
  - Don’t use malleable encryption without MAC
- **Should separate key between sub-protocols**
  - Similar issues also occurred in Threema[PST23], Matrix[ACDJ23]

[PST23] Paterson, Scarlata and Truong, “Three Lessons From Threema: Analysis of a Secure Messenger”, USENIX Security’23
[ACDJ23] *Albrecht*, *Celi, Dowling* and *Jones*, “Practically-exploitable Cryptographic Vulnerabilities in Matrix”, IEEE S&P’23
(Also, Black Hat Europe’22)

Step by step attack tracing

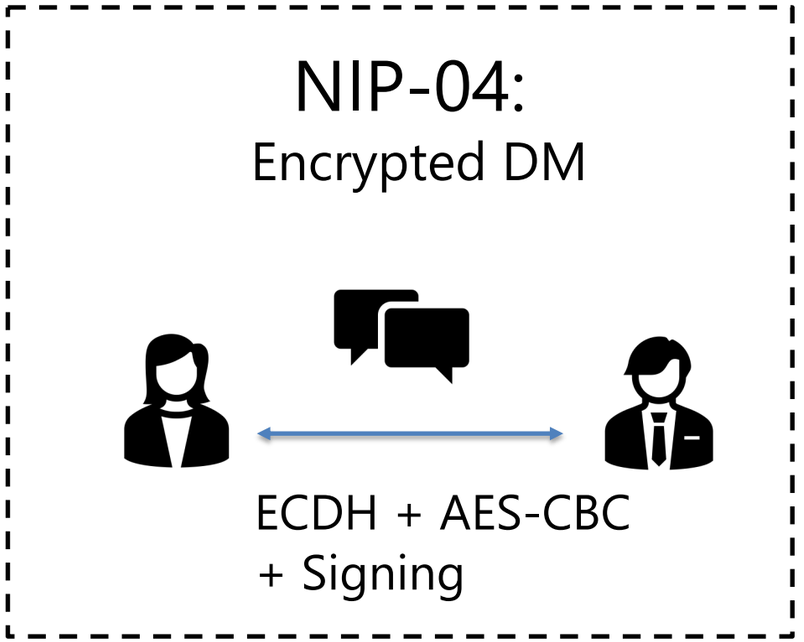
Breaking…

Plaintext integrity
(simple / cache poisoning)

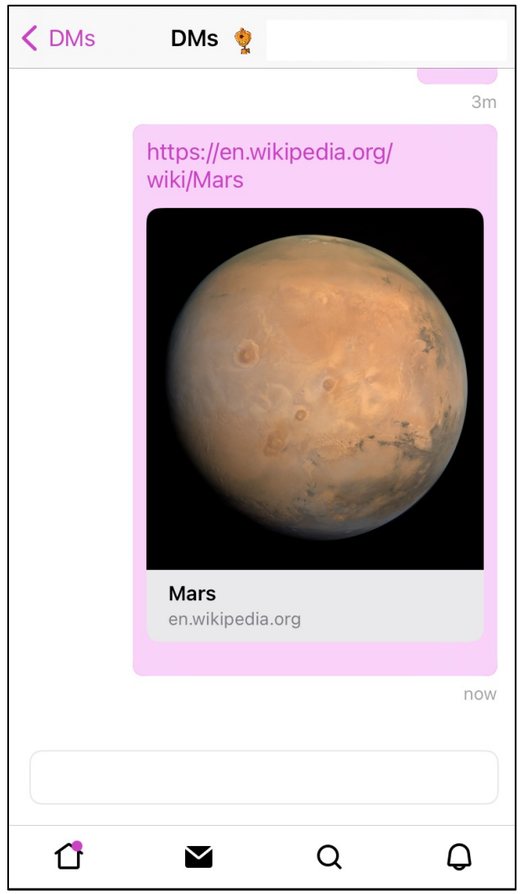
Ciphertext integrity

Ciphertext confidentiality

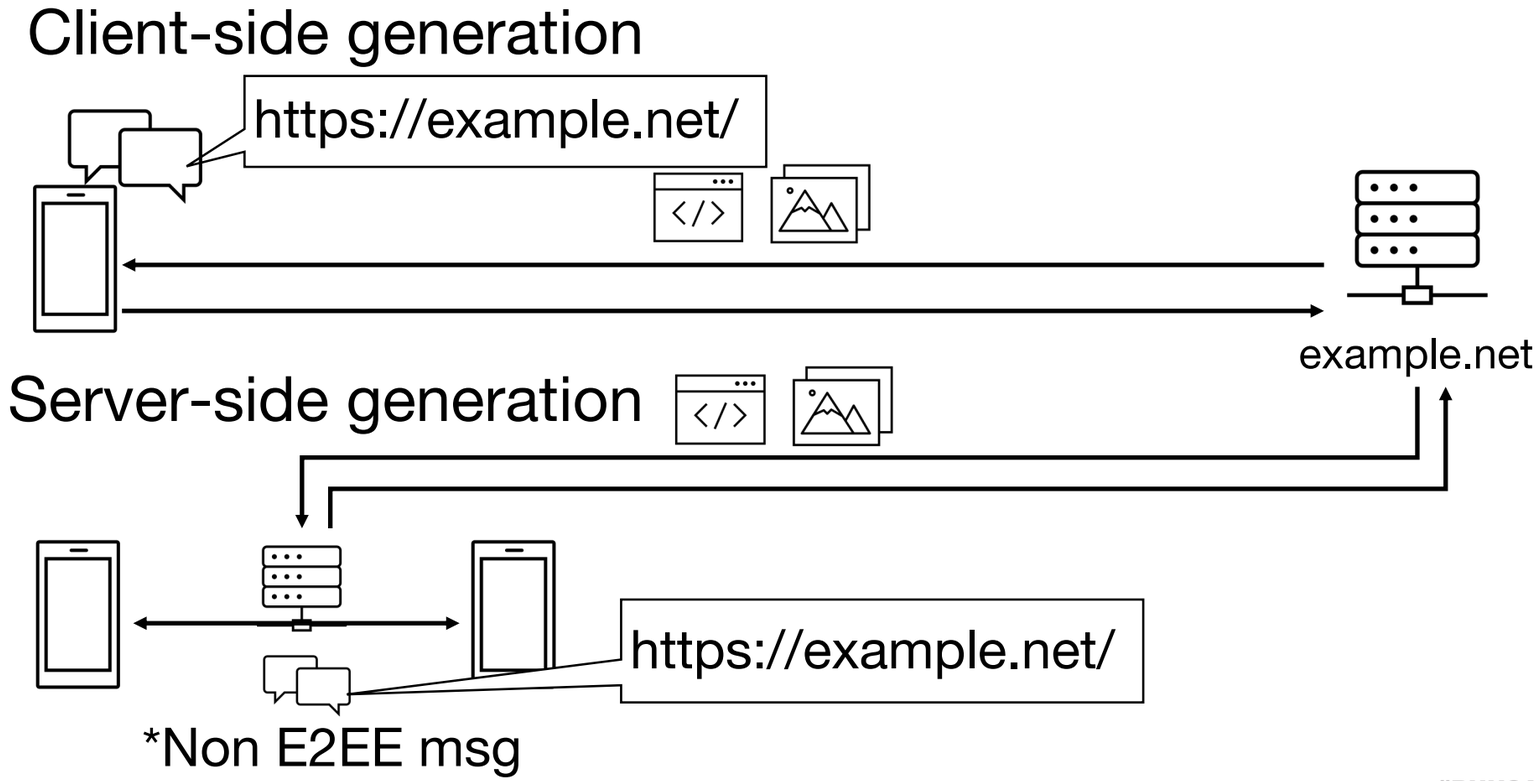
Remark: Encrypted Direct Messages specification (simplified)



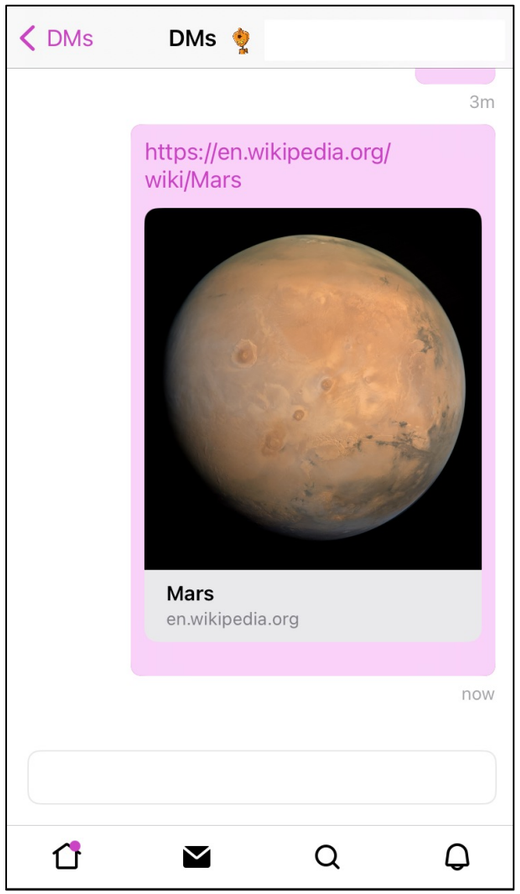
Link Preview in Messaging



- Automatically retrieves and displays elements from the webpage
  E.g., The webpage's title, part of its content, and images
- Someone must retrieve the page content (a sender, a receiver or a server)



Link Preview generation in **Encrypted** Messaging



Best practice

- Generate preview **ONLY** on the sender-side

Bad practice

- Generate preview on the receiver-side
- Known privacy issues (IP leakage): https://mysk.blog/2020/10/25/link-previews/

Is there any chance we can use it?

Many Nostr Clients

- Generate preview on the both sender-side and receiver-side

Thinking about plaintext recovery in the real-world **Encrypted** Messaging

- Hard to break cryptographic primitive standard
- But what if the recipient *helps* the attacker reveal an encrypted msg?
- How to win ? → Distinguishes & leaks decryption errors
  - Padding Oracle Attacks often appear in toy environments like CTFs

Q. Can we reproduce such an oracle in real-world systems?

Q. Can we reproduce such an oracle in real-world systems?

- Yes, we can! Receiver-side Link Preview generation helps us
- We finally find 3 attacks to break encrypted message confidentiality

URL recovery attack

Attacker's goal: disclose the authentication token in the URL
E.g., shared URL of cloud storage, web conference tools

https://us04web.zoom.us/j/[redacted]?pwd=[redacted]

Attacker wants to know

$$E_k(M) : M = \text{https://\{unknown domain\}/\{unknown part\}}$$

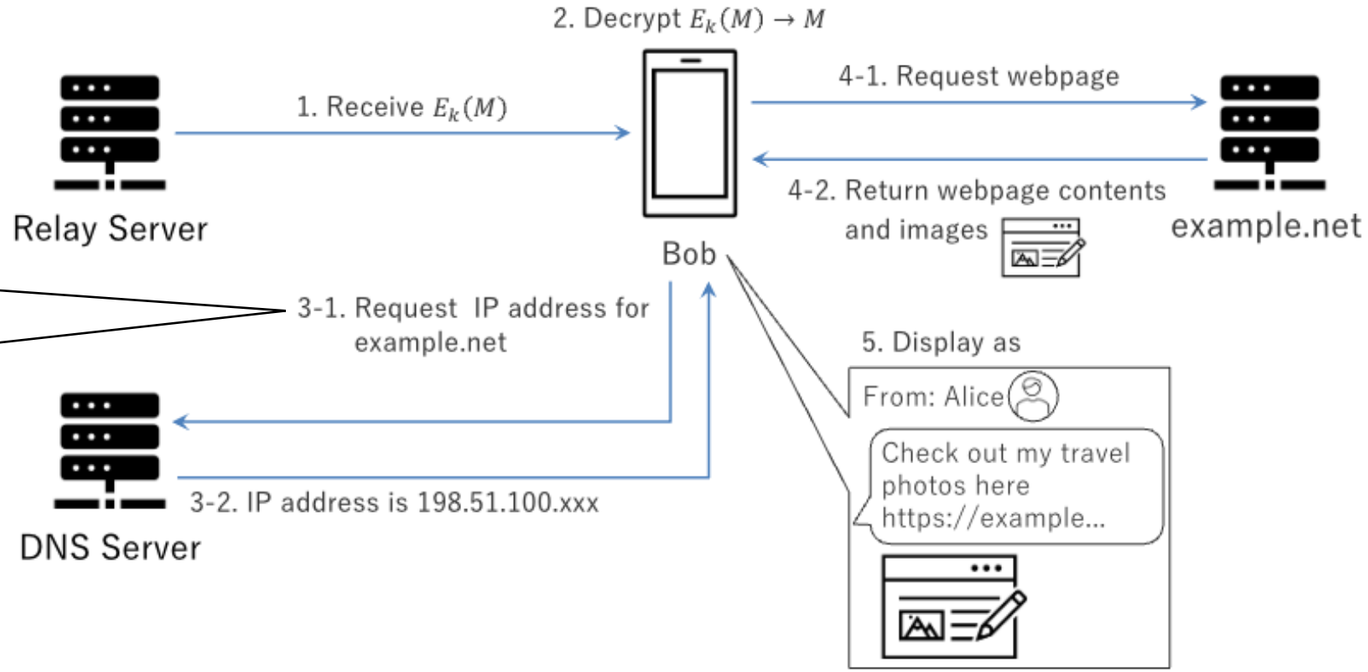
Authentication token

URL recovery attack

Disclose domain part

- Attacker can obtain domain part of URL via DNS or TLS SNI field
- Just by opening the message, DNS queries and TLS ClientHello packets are sent due to the automatic execution of link previews.

The attacker learned that
the domain part is “example.net”



URL recovery attack

Disclose authentication token

- Force the authentication token to be sent to the attacker's server
- Generate a modified ciphertext Ek(M′) where the domain is changed to a malicious one
- When the victim receives Ek(M′), the token is sent to the malicious URL via Link Preview

$E_k(M) : M =$ https://example.net/{unknown part}

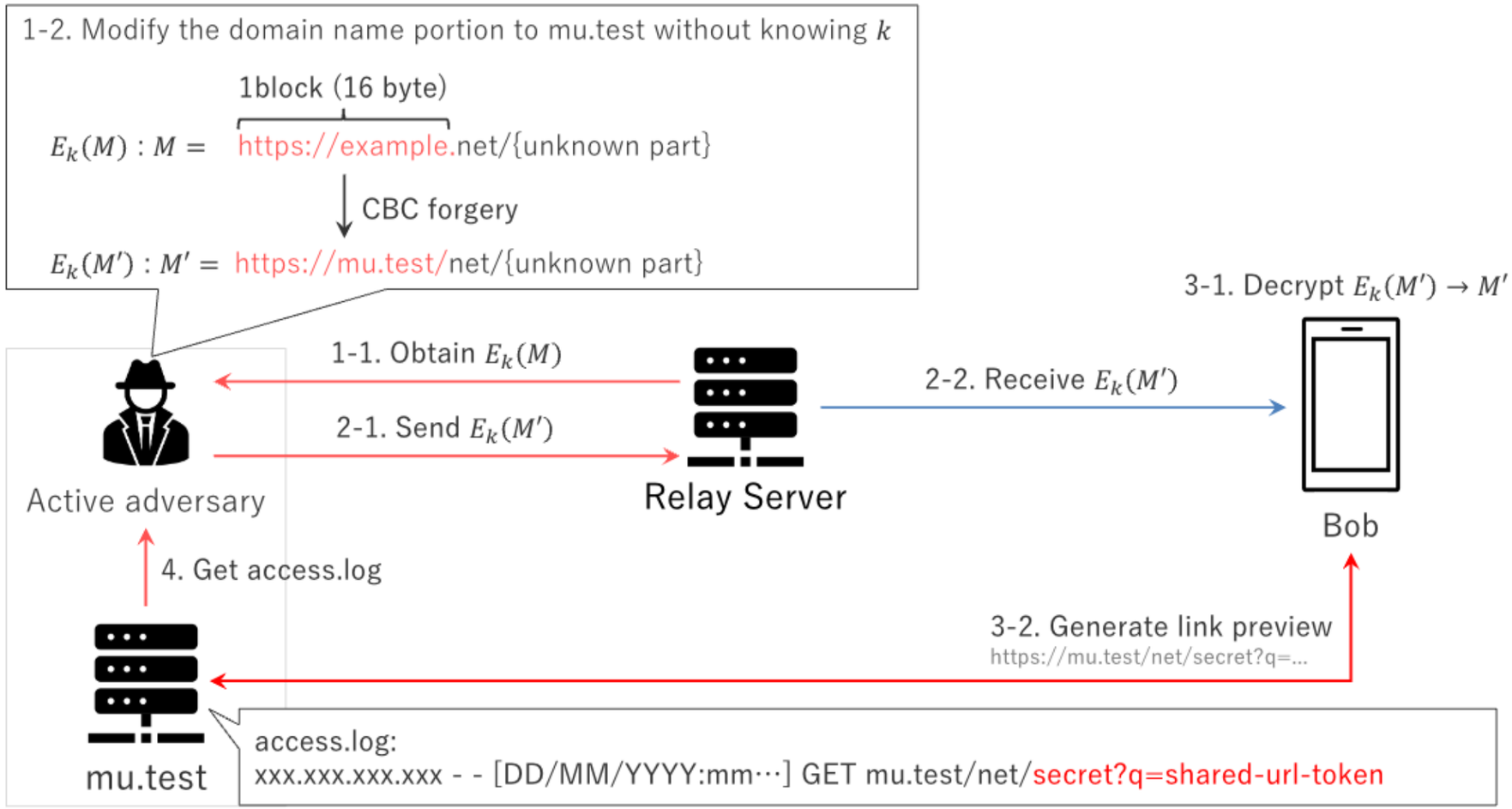
⇩ Encrypted DM forgery

$E_k(M') : M' =$ https://mu.test/net/{unknown part}

1Block (16Byte)

URL recovery attack

Disclose authentication token

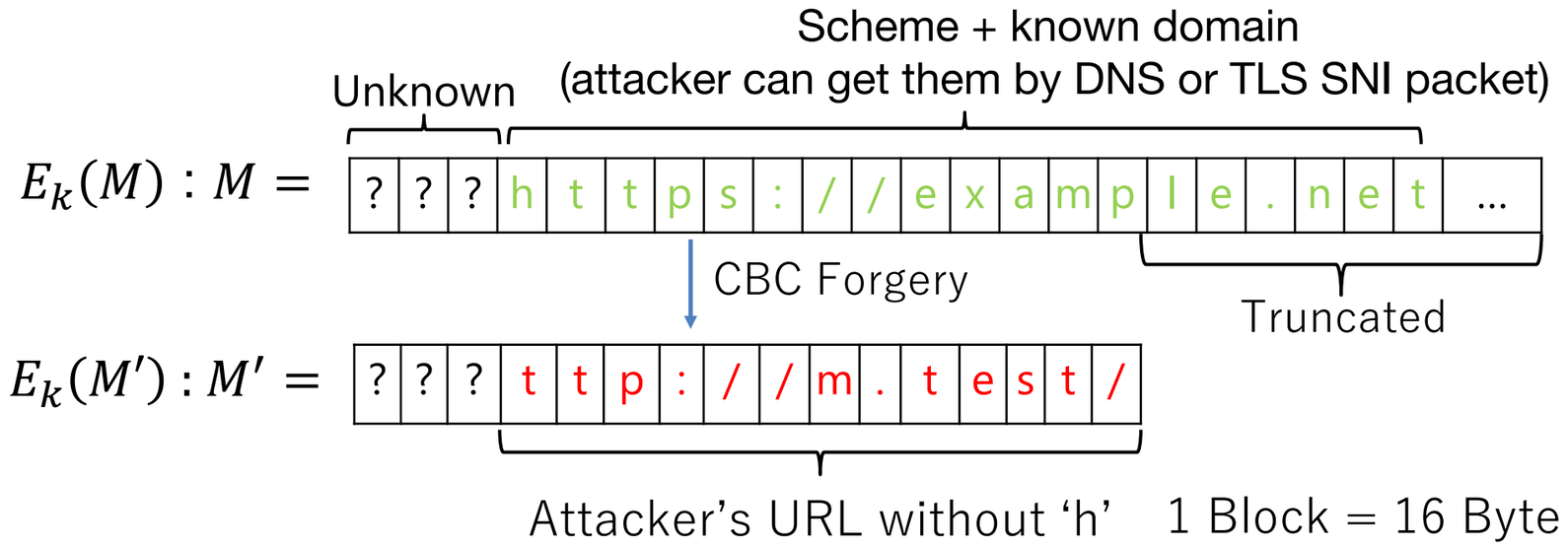


Link Preview Oracle Attack

Attack overview

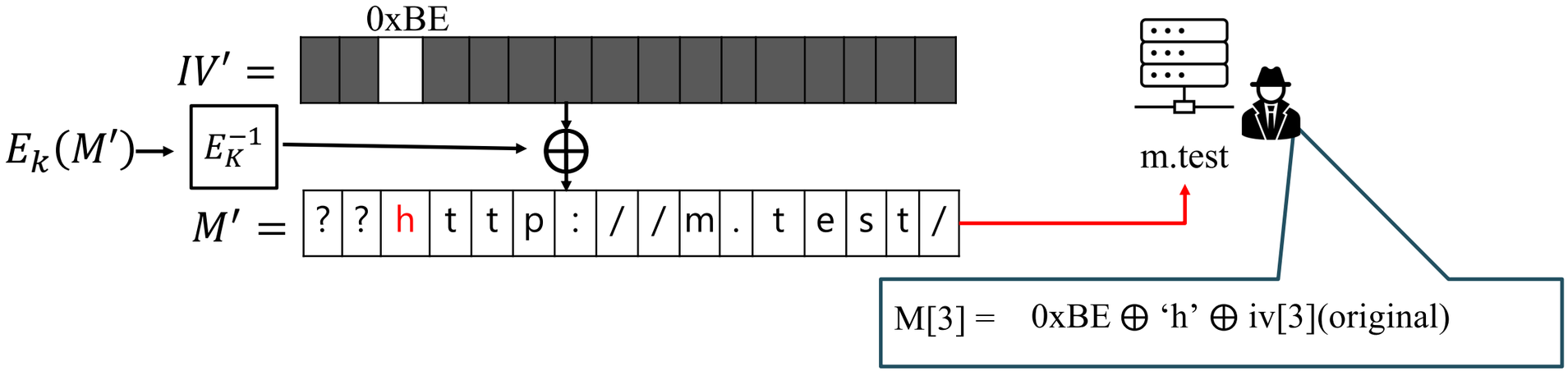
Attacker recover an encrypted message **before the encrypted URL.**
It works like a padding oracle attack.

Step1. Modify the encrypted via a CBC malleability,
producing a **partially attacker-controlled URL**



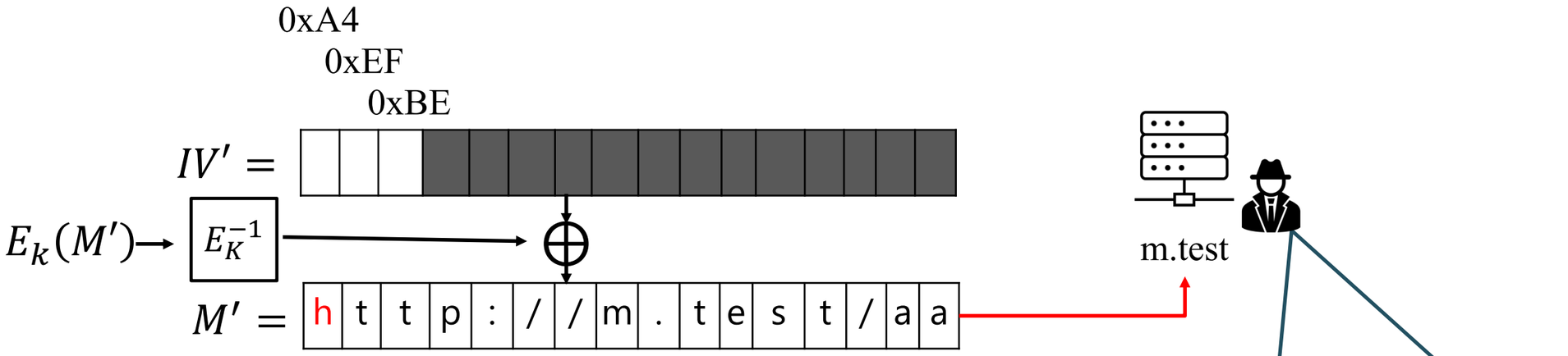
Link Preview Oracle Attack

Step2. Seek an IV′ such that the 3rd byte of M′ becomes **“h”.**
When “h” appears, the client fires a link preview, allowing the attacker to detect ‘h’ ← IV’[3](0xBE) ⊕ $E_K^{-1}(M')[3]$



Link Preview Oracle Attack

Step3. Repeat **Step 2** for the second and first bytes



M[3] = 0xBE ⊕ ‘h’ ⊕ iv[3](original)

M[2] = 0xEF ⊕ ‘h’ ⊕ iv[2](original)

M[1] = 0xA4 ⊕ ‘h’ ⊕ iv[1](original)

*Index starts with 1

Takeaway : Ciphertext Confidentiality

- **Remark: SHOULD use Authenticated Encryption (AE)**
  - E.g., AES-GCM, ChaCha20-Poly1305
  - Don’t use malleable encryption without MAC
- **SHOULD generate preview ONLY on the sender-side**

3 Takeaways : Whole of this presentation

1. Decentralized Architecture’s Untapped Risks and Rewards

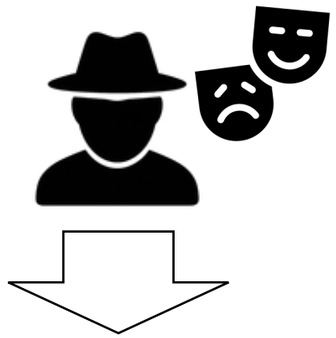
- Removing a central authentication server in Nostr brings new freedoms but also introduces subtle security pitfalls
- Multi-layered security are lost, and cryptographic weaknesses are immediately upgraded to practical attacks.

3 Takeaways : Whole of this presentation

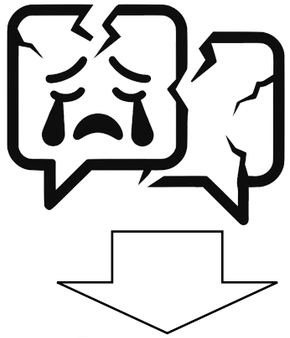
2. Hands-On Attacks & Immediate Mitigation

We guided our footsteps, and you learn how to destroy integrity & confidentiality

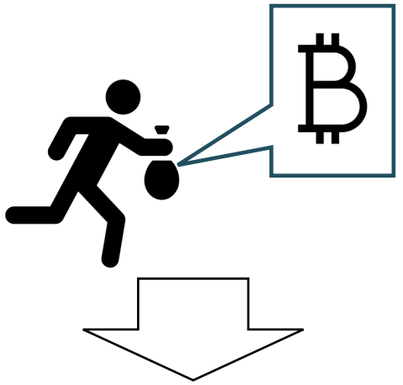
Identify the root cause and understand mitigation



- Signature verification Bypass



- Lack of key separation
- Receiver-side preview generation



- Verification Bypass

3 Takeaways : Whole of this presentation

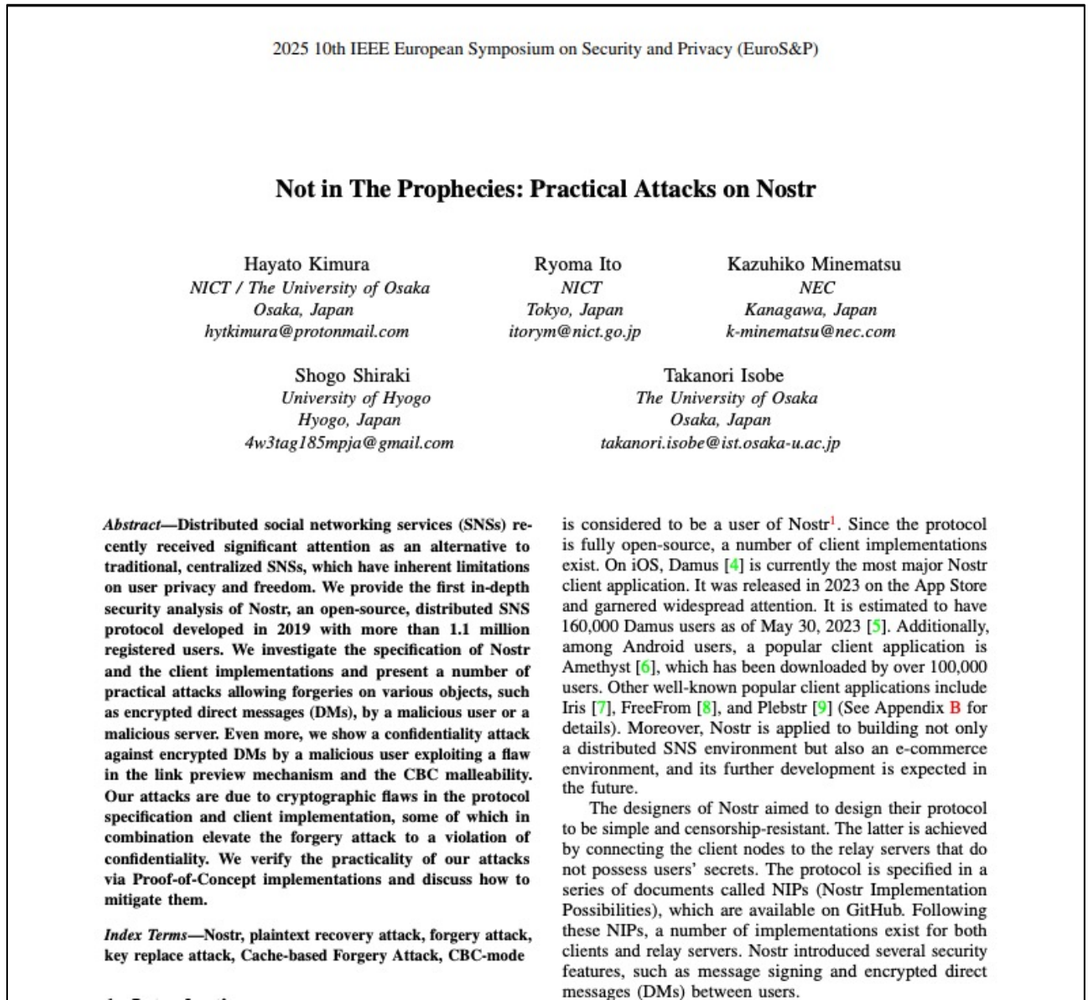
3. Blueprint for Future-Ready Decentralized Systems

Items	Nostr	Blueprint
Signature	Signing is mandatory. But there is no concrete specs for verifying.	Signing & verifying are mandatory
Link Preview	No specs (Mostly receiver-side generation)	Sender-side generation
Public Key Authenticity	No specs (NIP-05 Badge is available, but an authenticity is out of scope)	• Out-of-band authentication • Key Transparency

Summary

- First cryptographic deep-dive into Nostr, a distributed SNS.
- Find practical attacks caused by cryptographic & implementation flaw.
- Client is the trust anchor.
- Mandatory signature checks, key-separation, and AEAD.
- Responsible disclosure, and patches

Our Paper



2025 10th IEEE European Symposium on Security and Privacy (EuroS&P)

Not in The Prophecies: Practical Attacks on Nostr

Hayato Kimura
NICT / The University of Osaka
Osaka, Japan
hytkimura@protonmail.com

Ryoma Ito
NICT
Tokyo, Japan
itorym@nict.go.jp

Kazuhiko Minematsu
NEC
Kanagawa, Japan
k-minematsu@nec.com

Shogo Shiraki
University of Hyogo
Hyogo, Japan
4w3tag185mpja@gmail.com

Takanori Isobe
The University of Osaka
Osaka, Japan
takanori.isobe@ist.osaka-u.ac.jp

***Abstract*—Distributed social networking services (SNSs) recently received significant attention as an alternative to traditional, centralized SNSs, which have inherent limitations on user privacy and freedom. We provide the first in-depth security analysis of Nostr, an open-source, distributed SNS protocol developed in 2019 with more than 1.1 million registered users. We investigate the specification of Nostr and the client implementations and present a number of practical attacks allowing forgeries on various objects, such as encrypted direct messages (DMs), by a malicious user or a malicious server. Even more, we show a confidentiality attack against encrypted DMs by a malicious user exploiting a flaw in the link preview mechanism and the CBC malleability. Our attacks are due to cryptographic flaws in the protocol specification and client implementation, some of which in combination elevate the forgery attack to a violation of confidentiality. We verify the practicality of our attacks via Proof-of-Concept implementations and discuss how to mitigate them.**

***Index Terms*—Nostr, plaintext recovery attack, forgery attack, key replace attack, Cache-based Forgery Attack, CBC-mode**

is considered to be a user of Nostr[1]. Since the protocol is fully open-source, a number of client implementations exist. On iOS, Damus [4] is currently the most major Nostr client application. It was released in 2023 on the App Store and garnered widespread attention. It is estimated to have 160,000 Damus users as of May 30, 2023 [5]. Additionally, among Android users, a popular client application is Amethyst [6], which has been downloaded by over 100,000 users. Other well-known popular client applications include Iris [7], FreeFrom [8], and Plebstr [9] (See Appendix B for details). Moreover, Nostr is applied to building not only a distributed SNS environment but also an e-commerce environment, and its further development is expected in the future.

The designers of Nostr aimed to design their protocol to be simple and censorship-resistant. The latter is achieved by connecting the client nodes to the relay servers that do not possess users' secrets. The protocol is specified in a series of documents called NIPs (Nostr Implementation Possibilities), which are available on GitHub. Following these NIPs, a number of implementations exist for both clients and relay servers. Nostr introduced several security features, such as message signing and encrypted direct messages (DMs) between users.

https://crypto-sec-n.github.io/