Assertions and Tokens
+
Path tracing

SPIFFE/SPIRE

Sep/2022

Recap

- **Nested model:** Allows appending new assertions to existing tokens
- **Token construction:** Old model was causing a double encoding problem, solved by reformulating the token construction
- **Token path tracing:** Allows identifying all the hops the token has passed
- **Identification:** Different identification possibilities

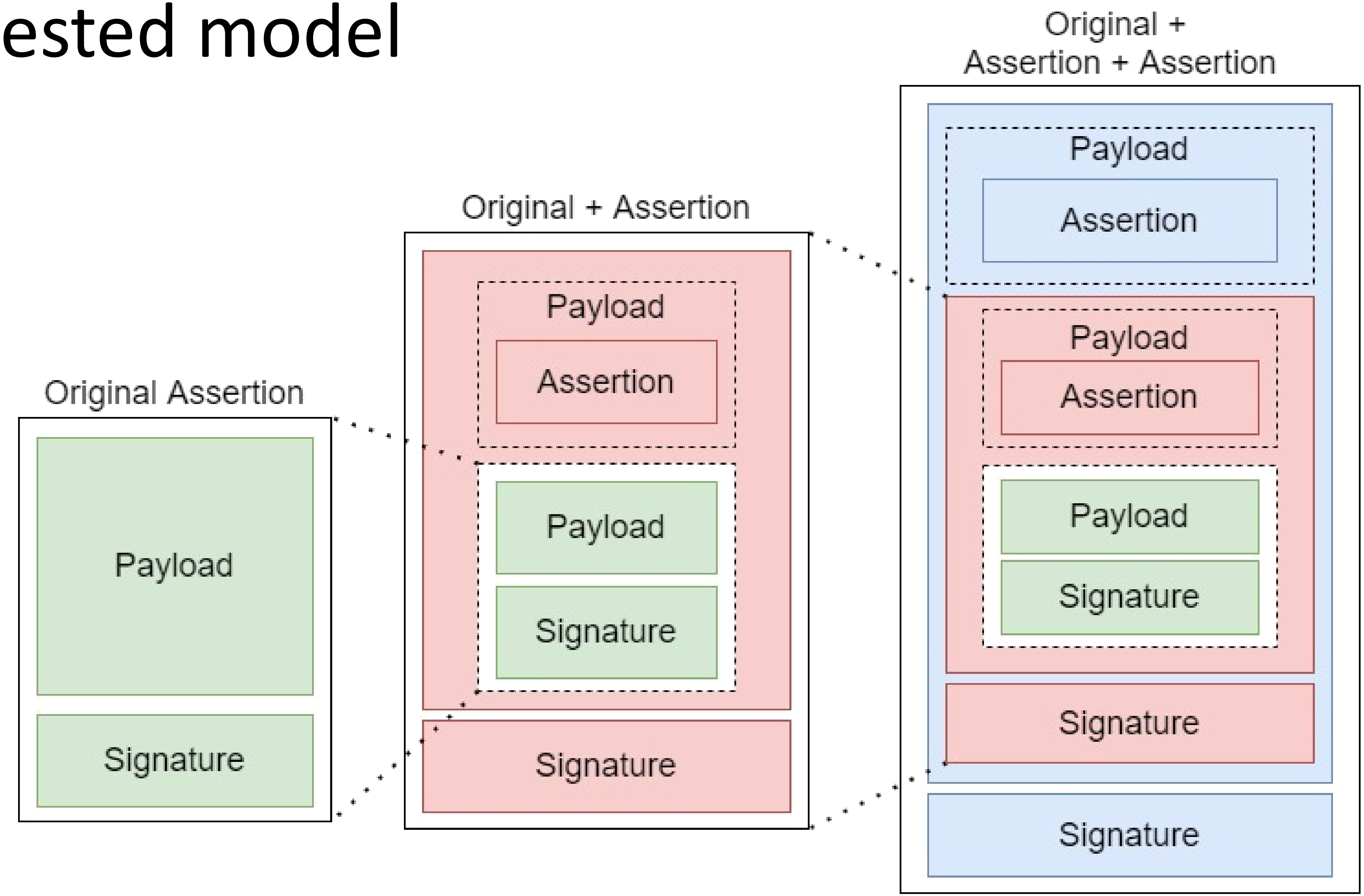
ID Possibilities

- **Anonymous mode:** Assertion issuer/audience are public keys with no ID reference – Biscuits-like
  - May use signature compression: Galindo-Garcia Signature for proving knowledge of precedent signatures

- **Cert-ID:** Assertion issuer/audience are a lightweight certificate containing ID details and public key

- **Directory Service:** Assertion issuer/audience are IDs used to retrieve certificates from a directory service

Nested model

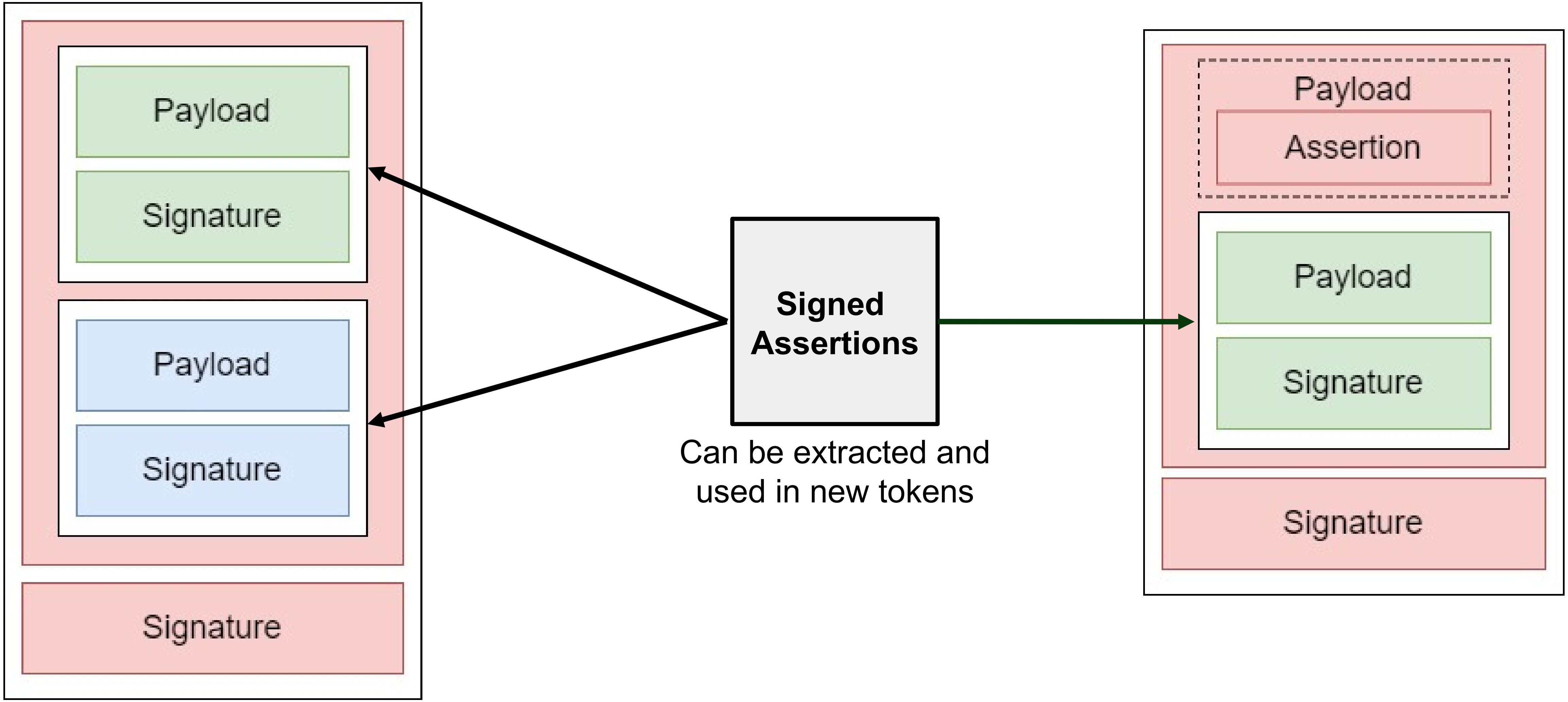


Assertion size comparision

Old model		
	SPIFFE-ID (Bytes)	**SVID** (Bytes)
x1	250	2.128
x2	520	4.988
x3	926	8.774
x4	1.466	13.848
x5	2.185	20.590
x6	3.143	29.594

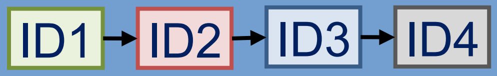
New model		
	SPIFFE-ID (Bytes)	**SVID (Bytes)**
x1	205	2.107
x2	412	4.216
x3	620	6.324
x4	827	8.434
x5	1.036	10.543
x6	1.244	12.653

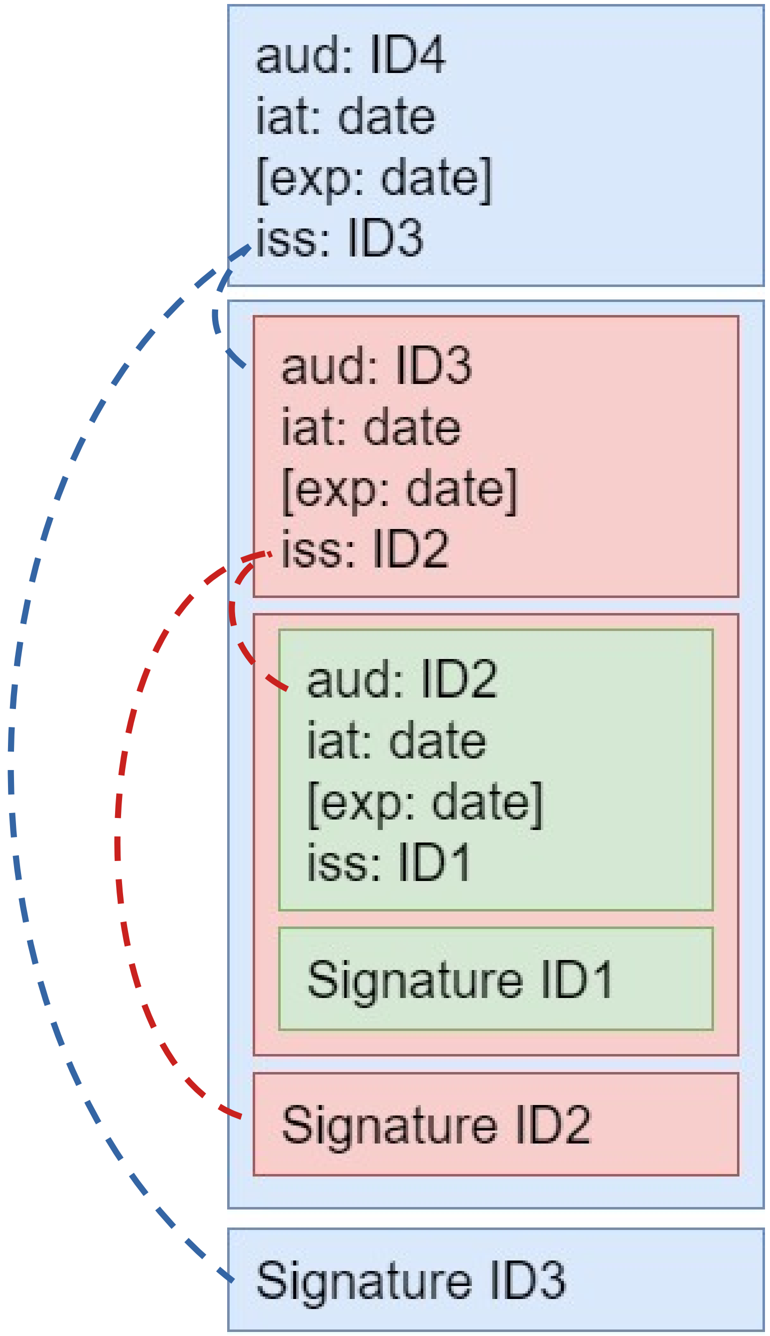
Group signed assertions



Token tracing

Link between issuer and audience





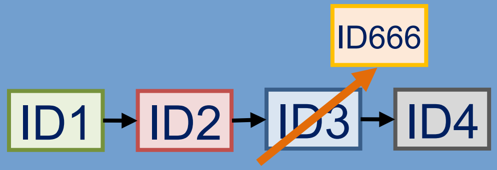
Attack Scenarios

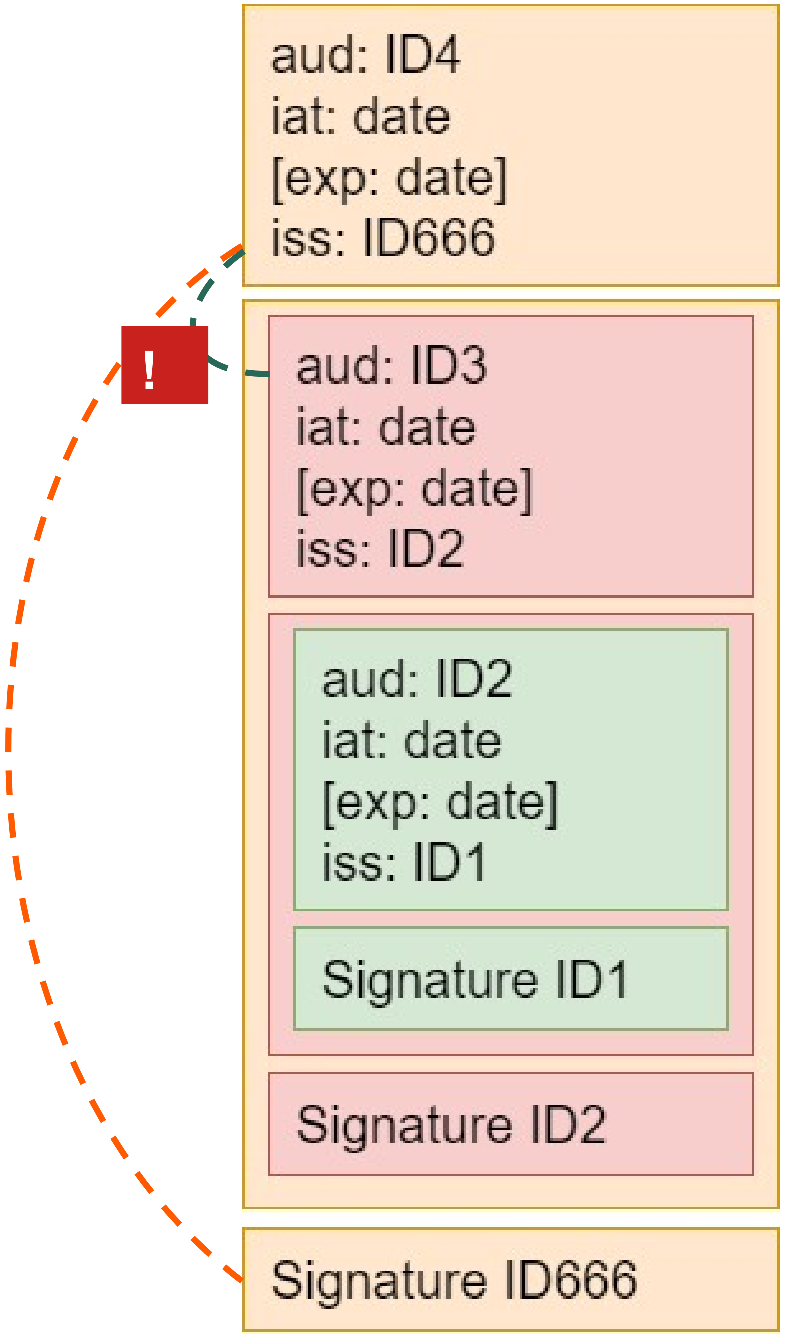
Attack 1

Replacement of last assertion

FAIL

issuer
bearer **!= audience**

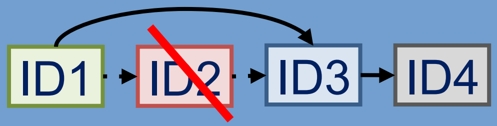


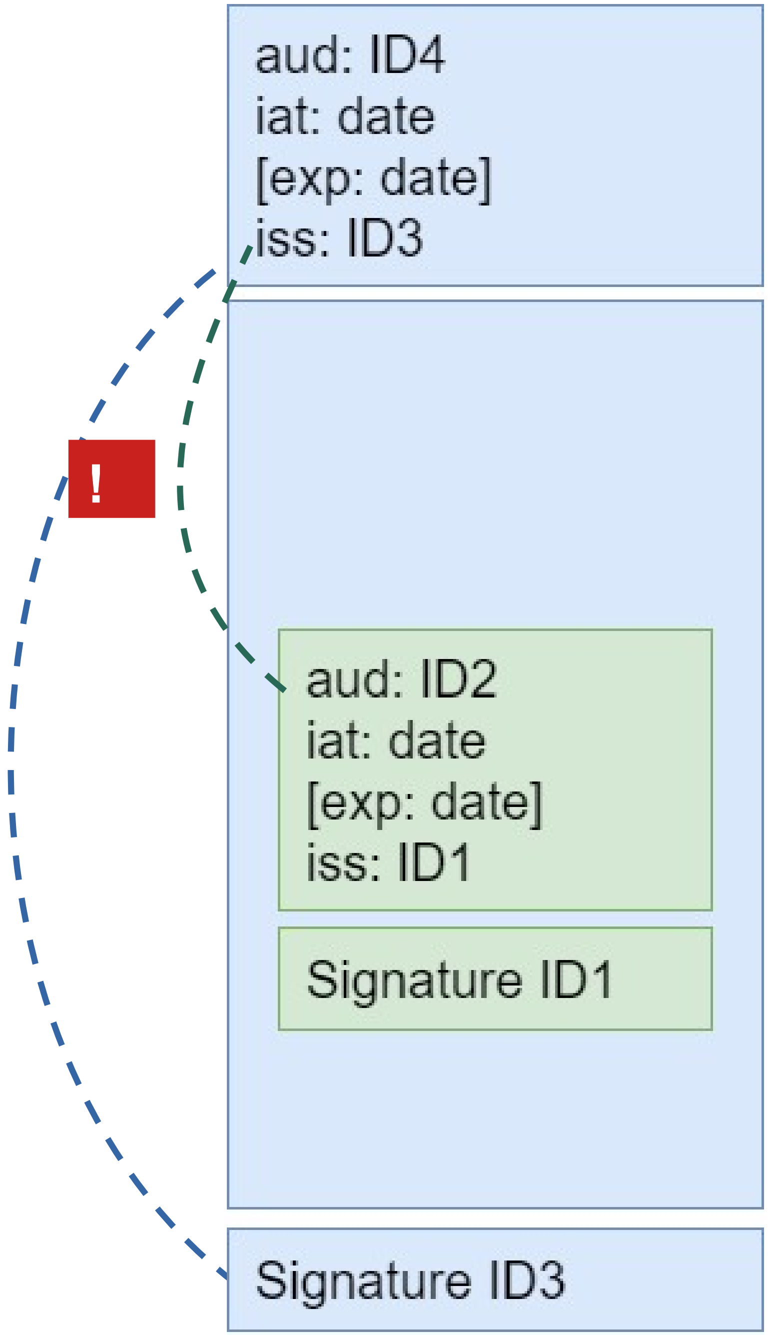


Attack 2

Removal of middle assertion

issuer
bearer != audience



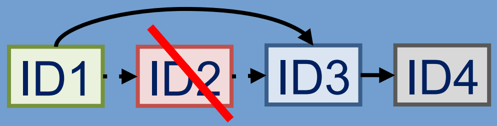


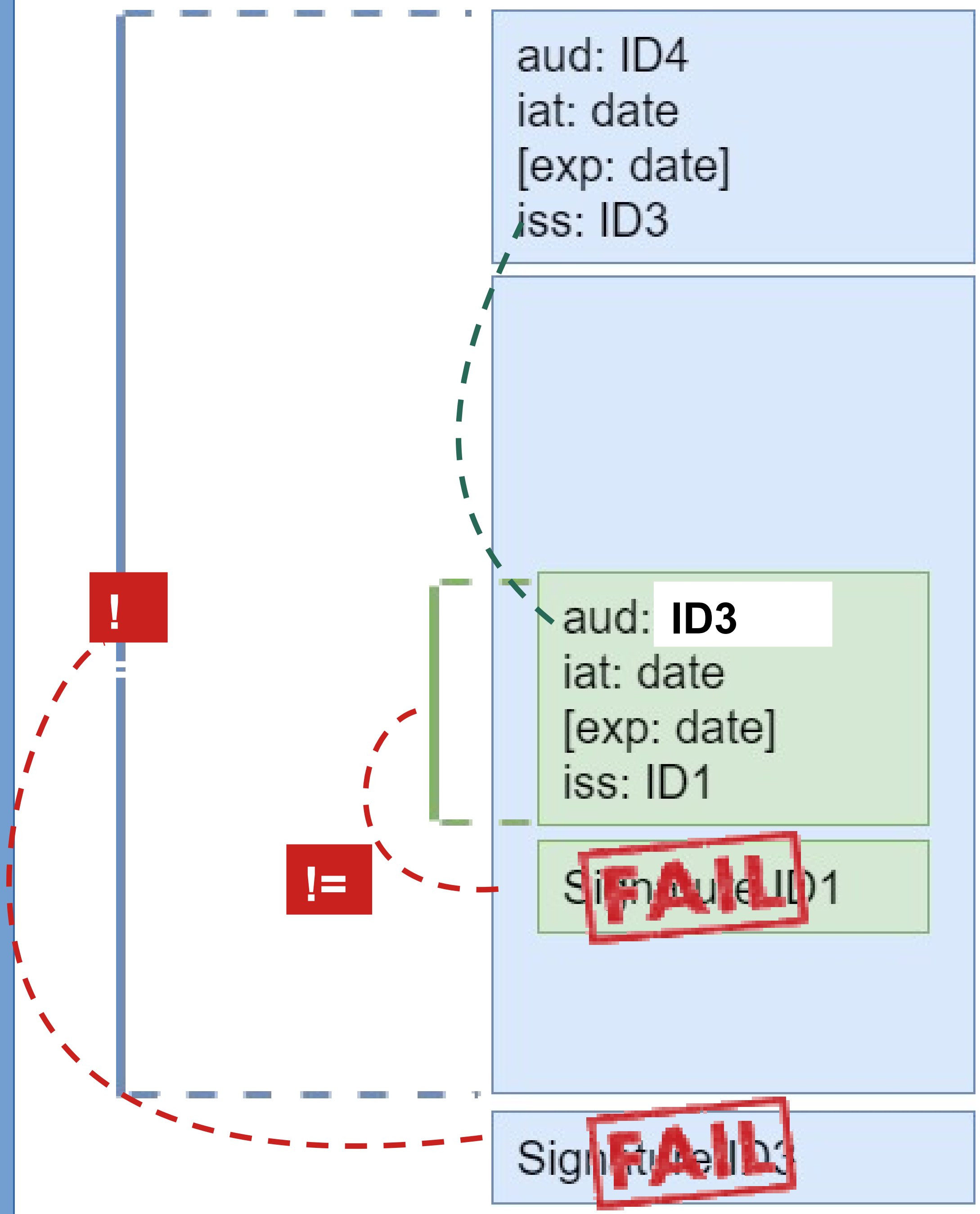
Attack 2

Token modification

FAIL

Hash chaining

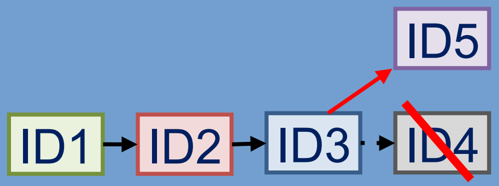


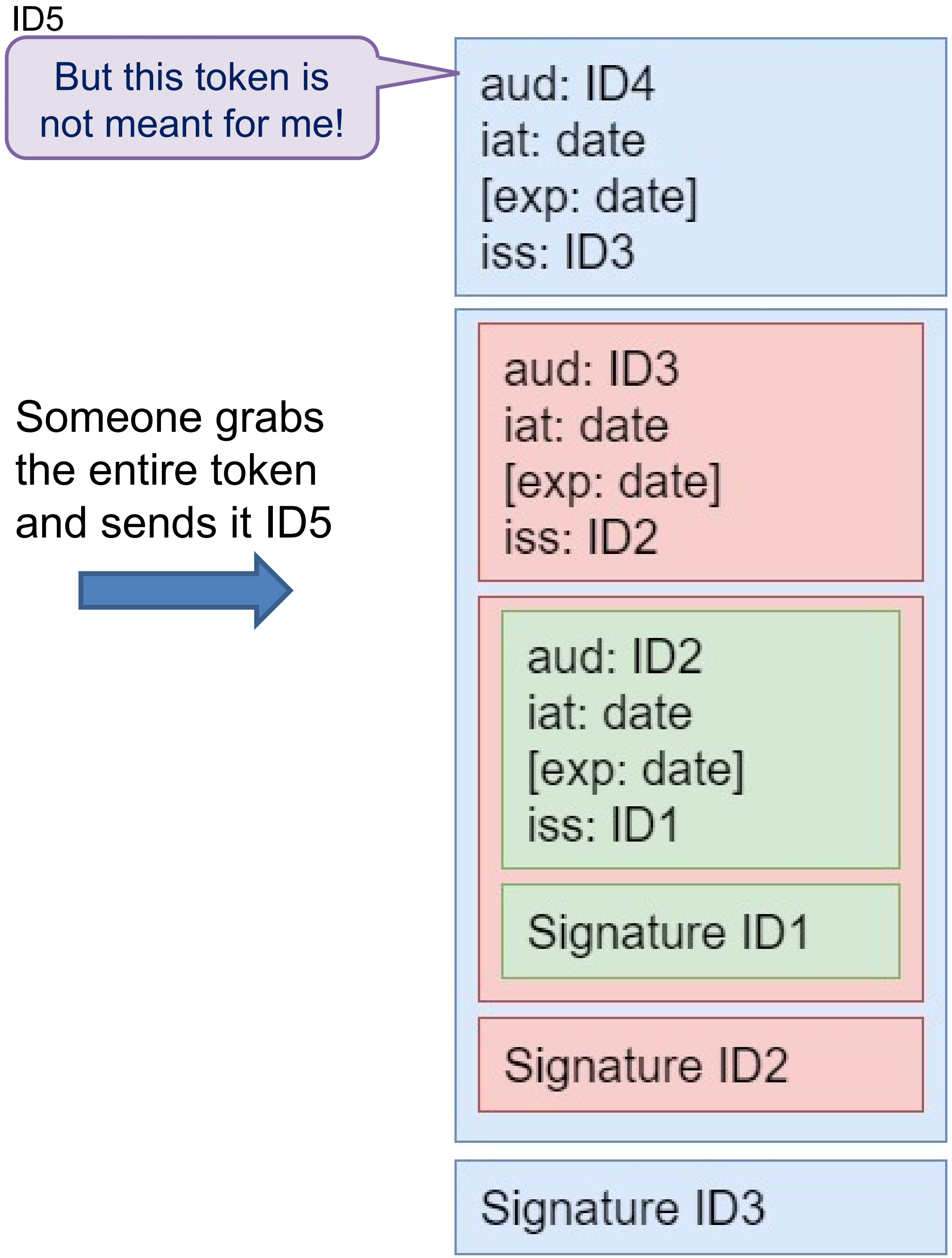


Attack 3

Identity misbinding

Audience verification

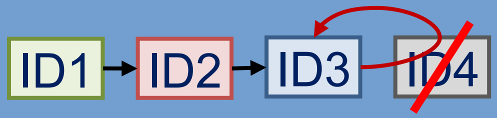




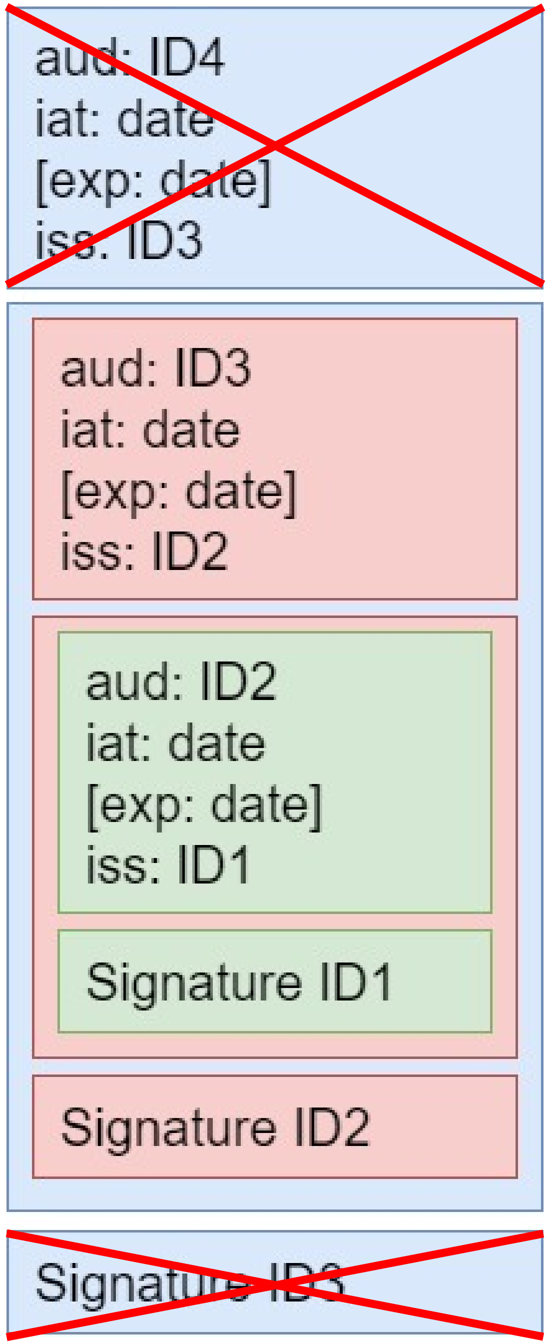
Attack 4

Identity loop

Some options...



Someone grabs old part of token and sends it back to ID3

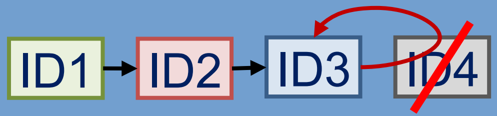


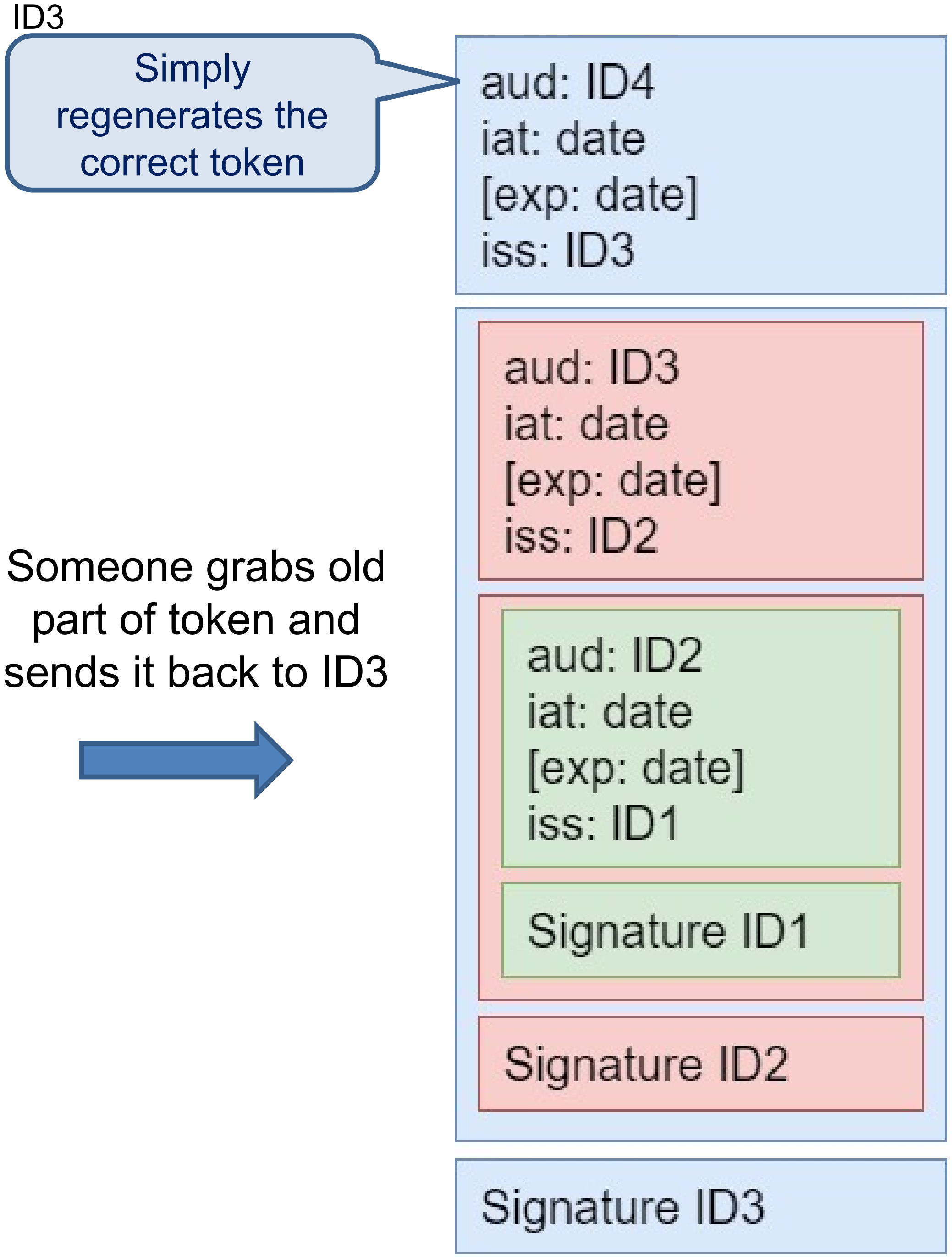
Attack 4

Identity loop

1: not an issue

(actually a feature: refresh token)



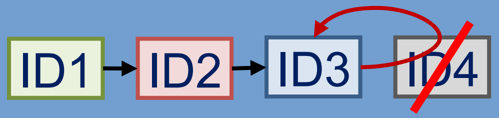


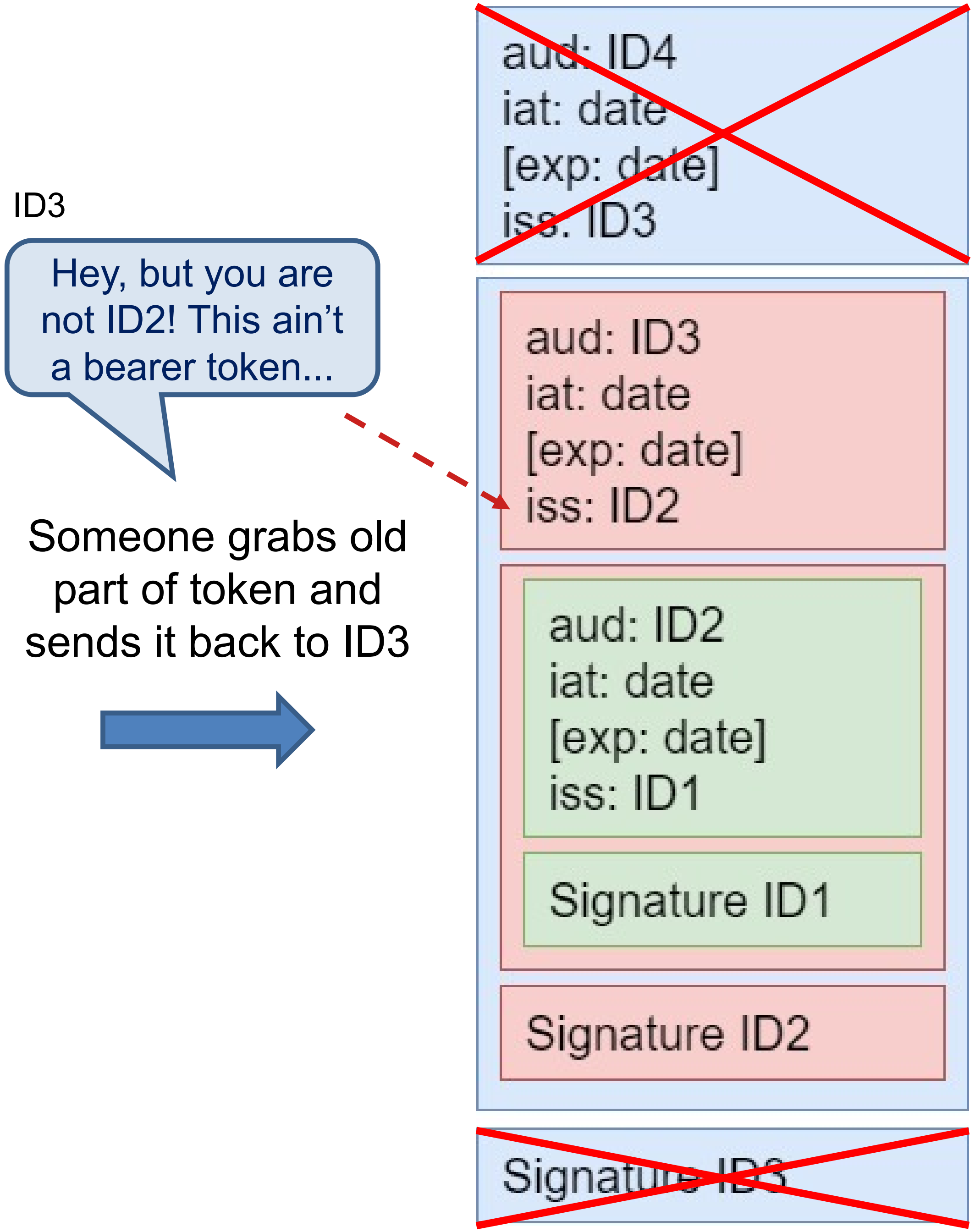
Attack 4

Identity loop

2: Issuer verification

(bad for proxies...)



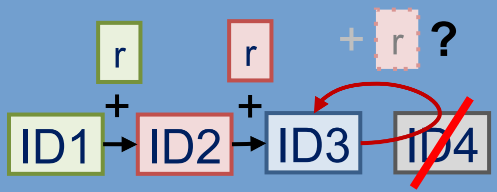


Attack 4

Identity loop

3: Loop protection

(freshness verification: challenge-response)



Hash-based:

- Challenge: hash(nonce)
- Response: nonce

  Con: increases tokens

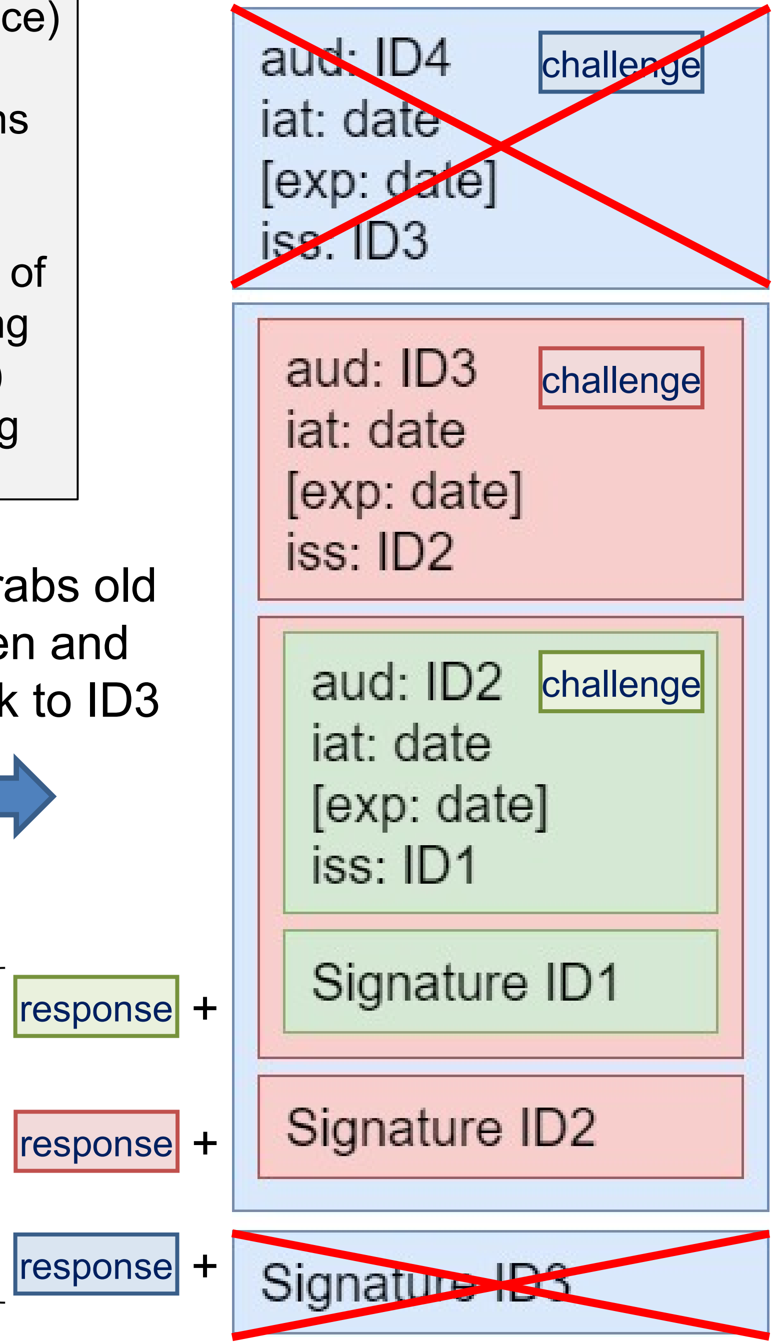
Signature-based

- Challenge: [implicit]
- Response: signature of whole token (including its internal signature)

➔ Con: extra processing and bandwidth

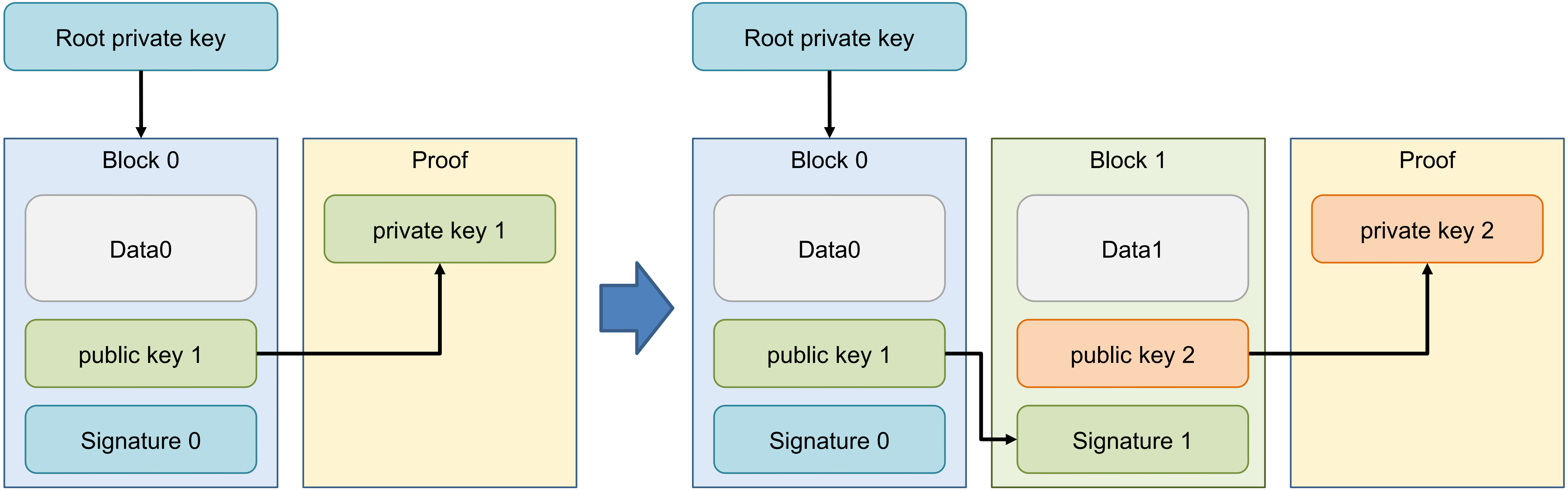
Someone grabs old part of token and sends it back to ID3

verified and then discarded by receiver



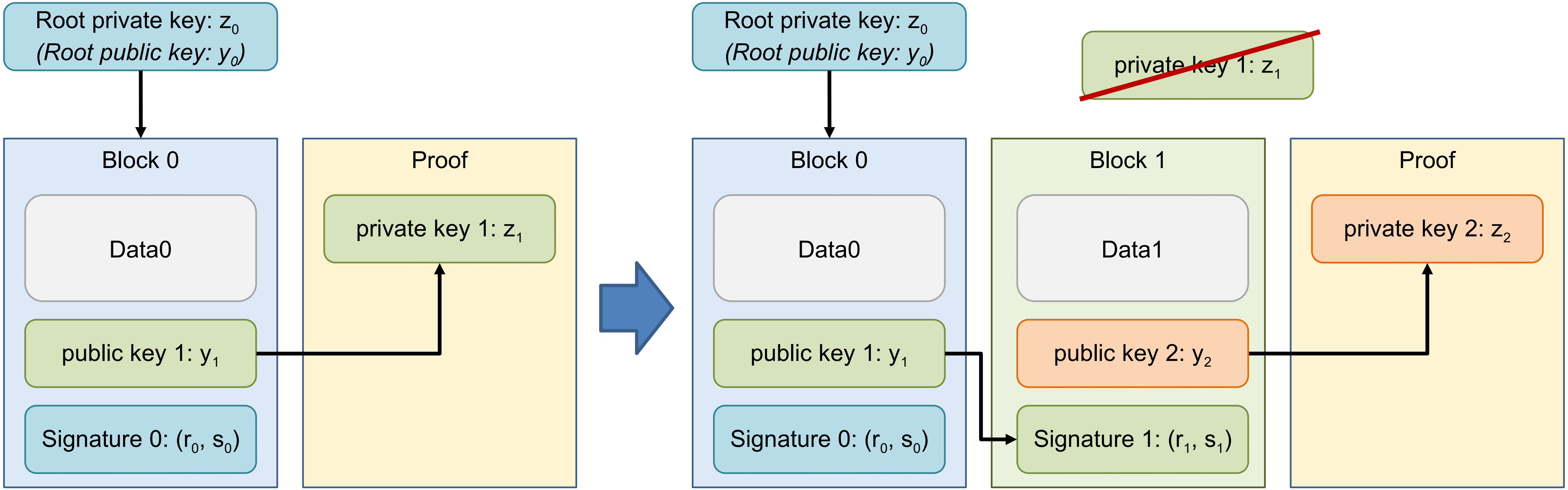
Signature Scheme

Biscuits



Root private key
Block 0
Data0
public key 1
Signature 0
Proof
private key 1
Root private key
Block 0
Data0
public key 1
Signature 0
Block 1
Data1
public key 2
Signature 1
Proof
private key 2

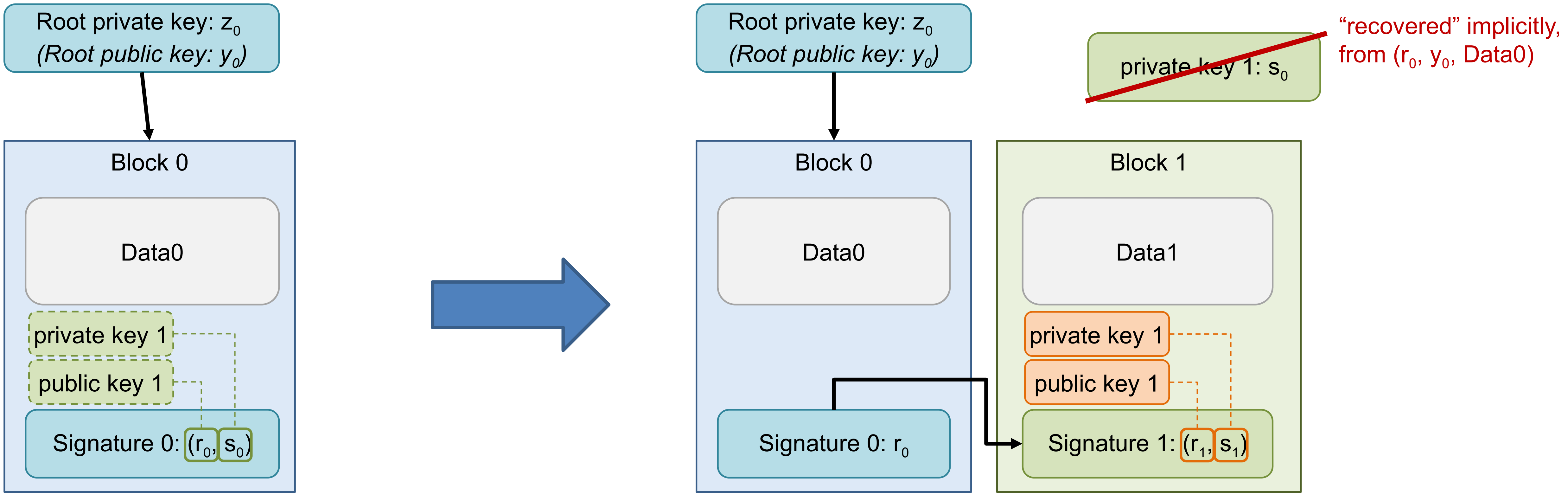
Biscuits (using Schnorr-based signatures, like EdDSA)



Schnorr-Sig (priv: z, pub: $y=g^z$)
$r = g^k$, k picked at random
h = Hash(r, Data)
$s = k + h \cdot z$
Output: (r, s)

Schnorr-Verif (priv: z, pub: $y=g^z$)
Input: (r, s)
h = Hash(r, Data)
Verify: $r \cdot y^h = g^s$?
[Note: $r \cdot y^h = g^k \cdot (g^z)^h = g^{k+h \cdot z} = g^s$]

Biscuits (using concatenated Schnorr-based signatures: Galindo-Garcia-style)



Schnorr-Sig (priv: z_0, pub: $y_0 = g^{z_0}$)
$r_0 = g^{k_0}$, k_0 picked at random
$h_0 = \text{Hash}(r_0, \text{Data0})$
$s_0 = k_0 + h_0 \cdot z_0$
Output: (r_0, s_0)

Schnorr-Sig (priv: s_0, pub: r_0)
$r_1 = g^{k_1}$, k_1 picked at random
$h_1 = \text{Hash}(r_1, \text{Data1})$
$s_1 = k_1 + h_1 \cdot s_0$
Output: (r_1, s_1)

Galindo-Garcia-Verif: Input (r_0, r_1, s_1)
$h_0 = \text{Hash}(r_0, \text{Data0})$, $h_1 = \text{Hash}(r_1, \text{Data1})$
Set: $y_1 = r_0 \cdot y_0^{h_0}$ [note: $y_1 = g^{s_0}$]
Verify: $r_1 \cdot y_1^{h_1} = g^{s_1}$? [note: regular Schnorr]

Concatenation of *‘n’* Schnorr signatures

Let: $k(n)$ = random

Inputs:

g = message

curve point

~~r(n-1),~~ $s(n-1)$ = prev. signature $s(n-1)$

Remove after sign

Signature creation

$r(n) = g * k(n)$

$h(n) = Hash(r(n) \,||\, message \,||\, pubkey(n))$

$s(n) = k(n) - h(n) * s(n-1)$

signature $= \{r(n), s(n)\}$

Galindo-Garcia verification of *'n'* signatures

Let: $r(n)$, $s(n)$ = signature n

g = curve point

$y0$ = public key 0

Inputs: [$r0$, $r1$, ..., $r(n)$],

[$h0$, $h1$, ..., $h(n)$],

$s(n)$, $y0$

Signature verification

Calculate:

$$y(n) = r(n-1) - y(n-1) * h(n-1)$$

Check:

$$g * s(n) = r(n) - y(n) * h(n)$$

Signature validation runtime

Token with 10 signatures	Std. Signature scheme	Galindo-Garcia based scheme
1	15,666	8,806
2	16,057	15,548
3	19,031	7,823
4	8,724	12,274
5	18,621	14,156
6	15,904	8,223
7	17,341	11,199
8	13,056	14,249
9	10,706	9,473
10	9,559	8,149
Average runtime	**14,467**	**10,990**

Galindo-Garcia: proof by induction to *'n'*

Let:

signature = {r, s}

h(n) = Hash(r(n)+message(n)+pubkey(n))

k(n) = random

y0 = pubkey 0

g = curve base point

Inputs:

[r0, r1, …, r(n)]

[h0, h1, …, h(n)]

s(n), y0

Function to proof:

y(n) = r(n-1) - y(n-1) * h(n-1)

Base: (n=1)

y(1) = r(0) - y(0) * h(0) ✓

Galindo-Garcia

Hipothesis: (n+1)

y(2) = r(1) - y(1) * h(1)

Verify:

g * s(2) = r(2) - y(2) * h(2)

Next Steps

- Script for automating PoC environment installation and configuration
- Update PoC to implement token tracing solution, allowing the validation of all hops
- Study the viability of Galindo-Garcia scheme generalization to “n” signatures
- Generate assertions from SPIRE selectors
- Protobuf / JSON analysis
- Add ECDSA/EDDSA support to Utoken
- General benchmarks

USP UDESC INSTITUTO FEDERAL Catarinense Hewlett Packard Enterprise