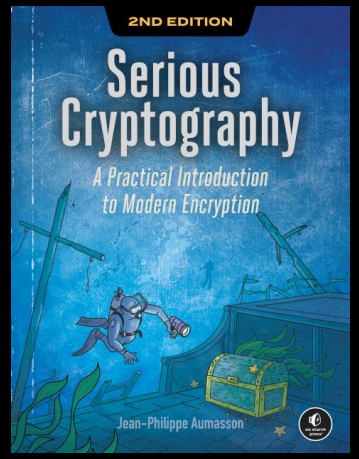

Hardening HSMs for
Banking-Grade Crypto Wallets
Black Hat 2024
JP Aumasson, Chervine Majeri

Whois

JP

- Taurus co-founder & CSO
- First BHUS talk was in 2013



Chervine

- Taurus lead research engineer
- First BHUS talk is now

TAURUS Crypto asset custody & issuance for banks (taurushq.com) regulated and running a marketplace for tokenized assets (t-dx.com)

In Geneva, Zurich, London, Paris, Vancouver, Dubai

Outline

1. **What is really an HSM?**
2. **Security and crypto internals**
3. **Attack surface and hardening**
4. **Best practices & a note on cloud HSMs**

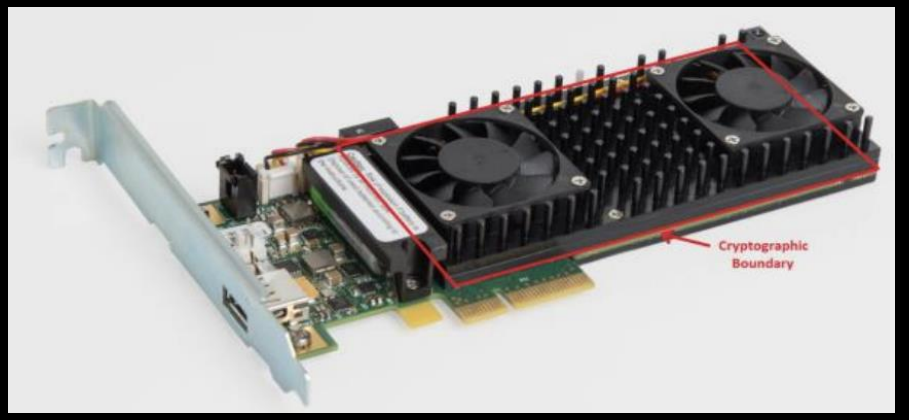
Disclaimer: This talk is based on our experience over 7 years with 3 HSM models, deployed in production in multiple environments. YMMV.

Hardware security module (HSM)

“A dedicated crypto processor that is specifically designed for the protection of the crypto key lifecycle” (HSM vendor)

Enterprise/cloud HSMs usually 1RU or PCIE card form factor

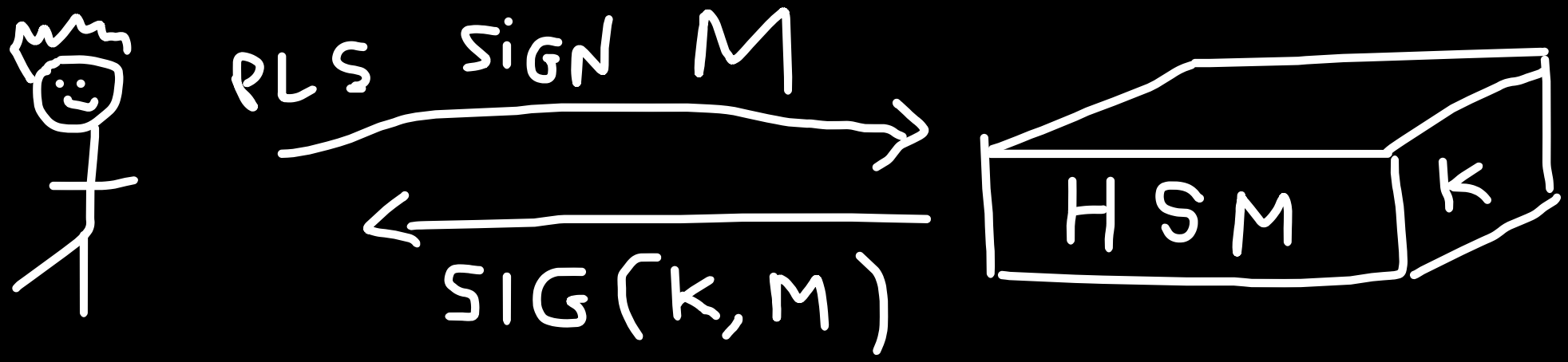
The actual HSM is the module in the appliance/card



HSM purpose

Store **secret keys** for crypto operations:

- Signature, decryption, symmetric encryption, MAC



High-assurance domain thanks to isolation & anti-tampering

Protect keys in case of servers/workstations compromise

HSM use case examples

- **Blockchain** transaction signing and TEE
- Code signing (HSM mandatory for MS Win apps)
- Database encryption/decryption (usually via KEKs)
- PKI root of trust (for CAs, enterprise PKIs, etc.)

https://www.flickr.com/photos/okolkman/22789012910/in/album-72157661146853781/

HSM interfaces

Crypto interface over **PCIe or USB**, **TCP/IP if** network-attached

Admin interface over serial port, **SSH**, **HTTP/REST** + **TLS**, **GUI**

```
The following top-level commands are available:

 Name                 (short)    Description
 ----------------------------------------------------
 client               c          > Client
 exit                 e          Exit [redacted] Shell
 help                 he         Get Help
 hsm                  hs         > Hsm
 my                   m          > My
 network              ne         > Network
 ntls                 nt         > Ntls
 package              pac        > Package
 partition            par        > Partition
 service              se         > Service
 status               st         > Status
 sysconf              sysc       > Sysconf
 syslog               sysl       > Syslog
 token                t          > Token
 user                 u          > User
 webserver            w          > Webserver
```

```
Syntax:      partition

The following subcommands are available:

 Name                 (short)    Description
 ----------------------------------------------------------------------------
 create               cr         Add Partition
 init                 i          > Init
 resize               resi       Resize Partition Storage Space
 rename               ren        Rename Partition
 createChallenge      createC    Create Crypto Officer or Crypto User challenge
 activate             a          Cache Partition PED key data
 deactivate           dea        Decache Partition PED key data
 list                 l          List Partitions
 show                 sh         Get Partition Information
 showContents         showC      Get Partition Objects
 showPolicies         showP      Get Partition Policies
 changePolicy         changePo   Set Partition Policy Value
 changePw             changePw   Set Partition Password
 delete               del        Delete Partition
 clear                cl         Delete Partition Objects
 backup               b          Backup Partition
 restore              rest       Restore Partition
 stcIdentity          st         Secure Trusted Channel Configuration
```

Security mechanisms (1/4)

- Local isolation (slots aka partitions)

HSM Partitions

HSM Partitions are independent logical HSMs that reside within the SafeNet HSM inside, or attached to, your host computer or appliance. Each HSM Partition has its own data, access controls, security policies, and separate administration access for at least some roles, independent from other HSM partitions (if your HSM supports more than one). Depending on the product, the HSM can contain multiple HSM partitions, and each partition can be associated with one or more Clients. Each HSM Partition has a special administrative account or role, who manages it.

Security mechanisms (2/4)

- Local isolation (slots aka partitions)
- RBAC, ABAC-ish model (with per-slot roles)

Administration Security Officer (ASO)

This user knows and can present the Admin Token SO PIN. The ASO's main role is to introduce the Administrator to the module. The following services are available to the ASO:

- Set the initial Administrator PIN value (ASO cannot change it later)
- Set the CKA_TRUSTED attribute on a Public object
- Set the CKA_EXPORT attribute on a Public object
- Exercise cryptographic services with Public objects
- Create, destroy, import, export, generate and derive Public objects
- Can change his/her own PIN

Administrator

This user knows and can present the Admin Token User PIN. The following services are available to the Administrator:

- Set or change Real Time Clock (RTC) value
- Read the System Event Log
- Purge a full System Event Log
- Configure the Transport Mode feature
- Specify the security policy of the HSM
- Create new SafeNet ProtectToolkit-C slots/tokens and specify their labels, SO PINs, and minimum PIN Length
- Initialize smart cards and specify their labels and SO PINs
- Destroy individual SafeNet ProtectToolkit-C slots/tokens
- Erase all HSM secure memory, including all PINs and User Keys
- Perform firmware upgrade operations
- Manage Host Interface Master Keys
- Exercise cryptographic services with Public objects on the Admin Token

Security mechanisms (3/4)

- Local isolation (slots aka partitions)
- RBAC model (with per-slot roles)
- PKCS#11 Cryptoki API

Bit Flag	Mask	Meaning
CKF_HW	0x00000001	True if the mechanism is performed by the device; false if the mechanism is performed in software
CKF_ENCRYPT	0x00000100	True if the mechanism can be used with **C_EncryptInit**
CKF_DECRYPT	0x00000200	True if the mechanism can be used with **C_DecryptInit**
CKF_DIGEST	0x00000400	True if the mechanism can be used with **C_DigestInit**
CKF_SIGN	0x00000800	True if the mechanism can be used with **C_SignInit**

5.9 Decryption functions

Cryptoki provides the following functions for decrypting data:

♦ **C_DecryptInit**

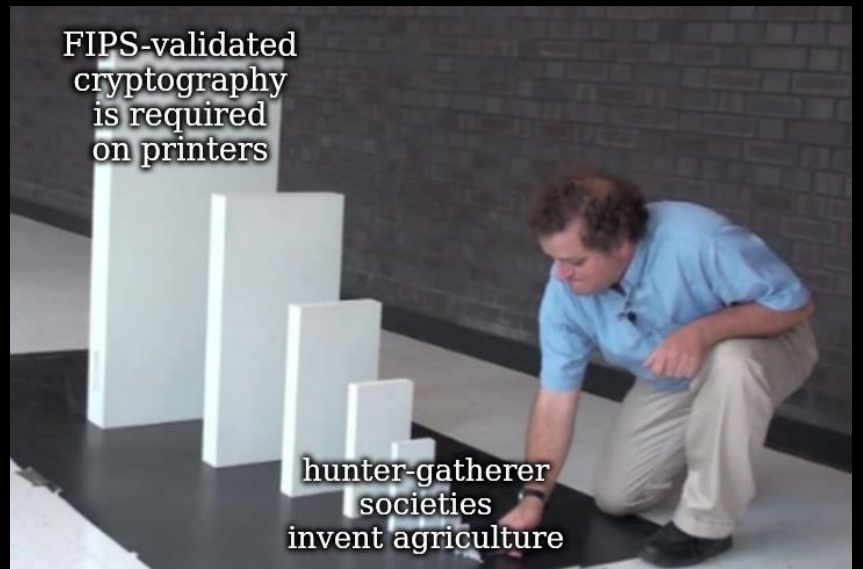
```
CK_DEFINE_FUNCTION(CK_RV, C_DecryptInit)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hKey
);
```

C_DecryptInit initializes a decryption operation. *hSession* is the session's handle; *pMechanism* points to the decryption mechanism; *hKey* is the handle of the decryption key.

The **CKA_DECRYPT** attribute of the decryption key, which indicates whether the key supports decryption, MUST be CK_TRUE.

Security mechanisms (4/4)

- Local isolation (slots aka partitions)
- RBAC model (with per-slot roles)
- PKCS#11 Cryptoki API
- FIPS 140-2/3 certified crypto and anti-tampering controls



Security mechanisms (5/4)

- Local isolation (slots aka partitions)
- RBAC model (with per-slot roles)
- PKCS#11 Cryptoki API
- FIPS 140-2/3 certified crypto and anti-tampering controls

May NOT include:

- Software exploit mitigations like ASLR and DEP
- Remote attestation mechanism

Internals overview (1/2)

- System-on-chip with a PPC core and crypto accelerators
- Some minimal Linux distrib, some bootloader
- Crypto software libraries
- Signed firmware updates

Libsodium	ISC (https://www.isc.or
Linux kernel	GPL-2.0 with exceptions
Linux-misc	GPL-2.0 (https://www.g
OpenSSL	OpenSSL (http://www.o
Protobuf	BSD-3-Clause "New" or "
Protobuf-c	BSD 2-clause "Simplified
Protocol Buffer Java Util Package	BSD-3-Clause "New" or "
U-boot	GNU GPL v2.0 only (http

```
23
24 # Start PCSC:
25 exec /sbin/pcscd
26
27 # Launch the HSM app, and wait for it to die.
28 exec /sbin/HSM
29
30 sendlog_text "LOG(CRITICAL)  HSM crashed:"
31
32 # Tell host that HSM quit:
33 sendlog_text "HSM terminated"
```

Internals overview (2/2)

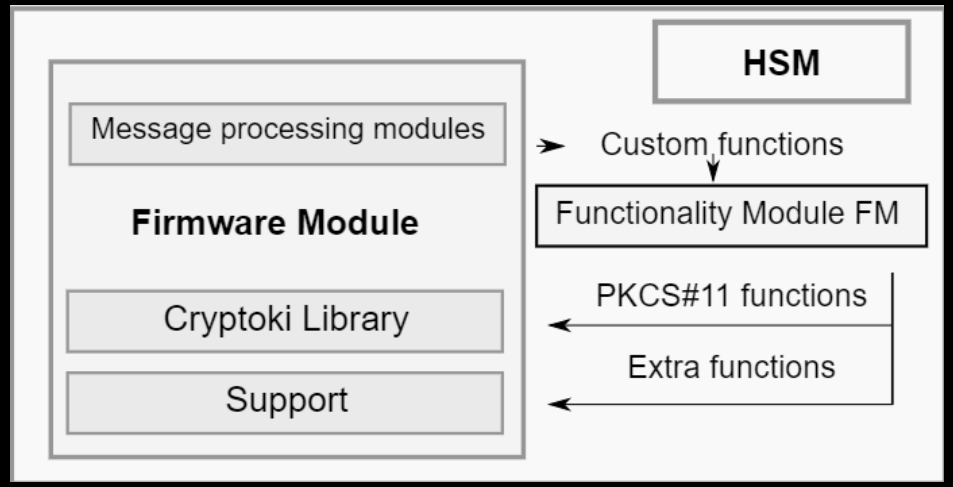
- Crypto support: mainly FIPS incl. legacy algorithms
- “True RNG” seeding a NIST 800-90A DRBG

- Full Suite B support
- Asymmetric: RSA, DSA, Diffie-Hellman, Elliptic Curve Cryptography (ECDSA, ECDH, Ed25519, ECIES) with named, user-defined and Brainpool curves, KCDSA and more
- Symmetric: AES, AES-GCM, DES, Triple DES, ARIA, SEED, RC2, RC4, RC5, CAST, and more
- Hash/Message Digest/HMAC: SHA-1, SHA-2, SM3 and more
- Key Derivation: SP800-108 Counter Mode
- Key Wrapping: SP800-38F
- Random Number Generation: designed to comply with AIS 20/31 to DRG.4 using HW based true noise source alongside NIST 800-90A compliant CTR-DRBG

Custom modules

- Firmware extension software component loaded by users
- Replace the original firmware's init()
- Must be developed C, using the vendor's SDK
- Size limitation (ex: 8MB)

```c
static int handle_init(Message_Handle token, Messages__HSMRequest *request) {
    if(request->checksum.data == NULL || request->checksum.len != 32) {
        return pack_reply(token, "invalid checksum", MESSAGES__HSMRESPONSE__CODE__Failure, NULL, 0, 0, NULL, NULL);
    }
    if(request->init == NULL || request->init->length == 0 || request->init->data.data == NULL || request->init->data.len == 0) {
        return pack_reply(token, "no init payload", MESSAGES__HSMRESPONSE__CODE__Failure, NULL, 0, 0, NULL, NULL);
    }
    uint32_t len = request->init->length;
    if(len > 1000000) {
        return pack_reply(token, "invalid total size: max 1000000 bytes", MESSAGES__HSMRESPONSE__CODE__Failure, NULL, 0, 0, NULL, NULL);
    }
    if(request->init->data.len > len) {
        return pack_reply(token, "invalid payload: longer than total size", MESSAGES__HSMRESPONSE__CODE__Failure, NULL, 0, 0, NULL, NULL
    }
```



What could go wrong (1/3)

- Compromised caller creds = free HSM requests (no filtering)
- PKCS#11 intrinsic flaws and limitations (see Ledger's paper)
- Bugs in the PKCS#11 implementation and HSM runtime

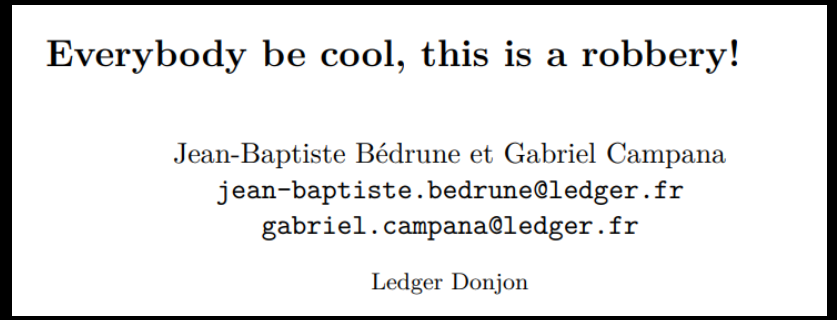
Yubico YubiHSM PKCS#11 Library Vulnerability (CVE-2023-39908)

Aug 14, 2023 • Christian Reitter
ID: CVE-2023-39908

Related articles: Yubico libykpiv Vulnerabilities II • Yubico yubihsm-shell Vulnerability (CVE-2021-43399) • Yubico libyubihsm Vulnerabilities (CVE-2021-27217, CVE-2021-32489)

Heiko Schäfer discovered a new security issue in the Yubico `yubihsm_pkcs11.so` driver library, which we disclosed together to Yubico. The YubiHSM PKCS#11 client-side library is designed to interact with Yubico HSM2 hardware security modules. Due to flaws in the memory handling, the library code accidentally returns 8192 bytes of previously used process memory under some circumstances. This impacts the memory confidentiality of the calling program for some usages.

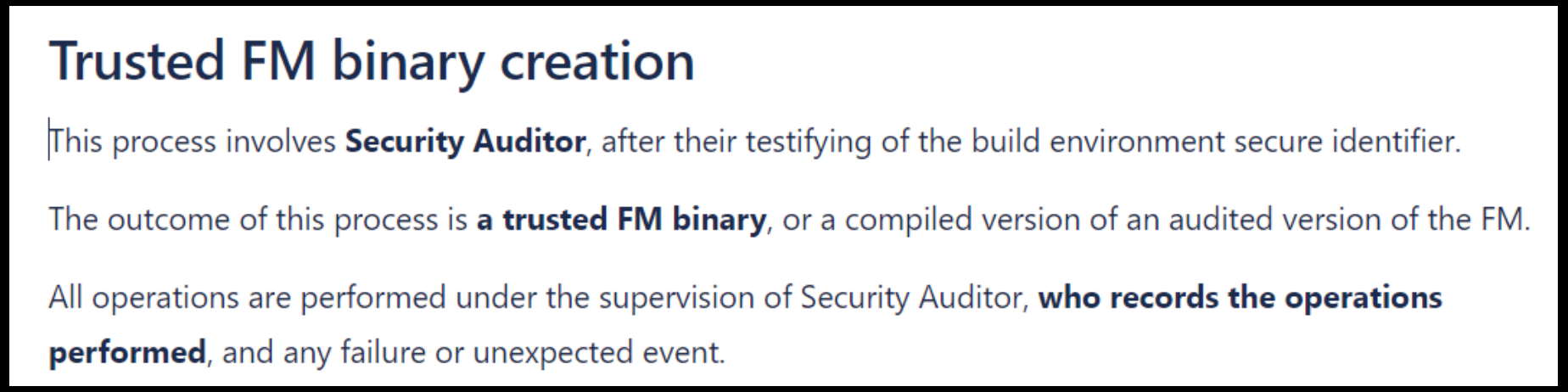
https://blog.inhq.net/posts/yubico-yubihsm-pkcs-vuln/



SSTIC 2019

What could go wrong (2/3)

- Knowns bugs in outdated OSS components (regreSSHion?)
- Cross-slot attacks (DoS, info leak, code exec?)
- Malicious custom module / supply-chain issues
- RNG issues (remember ROCA?)



Trusted FM binary creation

This process involves **Security Auditor**, after their testifying of the build environment secure identifier.

The outcome of this process is **a trusted FM binary**, or a compiled version of an audited version of the FM.

All operations are performed under the supervision of Security Auditor, **who records the operations performed**, and any failure or unexpected event.

What could go wrong (3/3)

Various HSM bugs:

- Remove a directory from the FS crashes if the name ands with “/”
- Logging "too much" (1 log per message) freezes the HSM freeze, needing a power-cycle
- Client-side segfaults with certain ECC crypto interfaces
- Inconsistent crypto interface between firmware versions

HSM hardening

A quick tour of measures proposed to harden HSMs

- Deployed in production
- Known tricks for “power users”
- Most won’t work with cloud HSMs

1/6: Attack surface reduction

- PKCS#11 API override, to only allow “authorized” usage/args
- Use directly the filesystem (rather than PKCS#11 objects)

```
FM_RV Startup() {
    CprovFnTable_t *fn_table = OS_GetCprovFuncTable();
    if(fn_table == NULL) {
        return 2;
    }
        fn_table->C_GetAttributeValue = GetAttributeValueOverride; //GetAttribute because it can be used to ge
        fn_table->C_WrapKey=WrapKeyOverride; //WrapKey because we don't want wrapped versions getting out
        fn_table->C_DeriveKey=DeriveKeyOverride; //DeriveKey because BIP32
        fn_table->CT_CopyObject=CTCopyObjectOverride; //CopyObject because if Sensitive is missing, one could
        fn_table->C_CopyObject=CopyObjectOverride; //Same but not across sessions
    fn_table->C_SignInit=SignInitOverride; // Prevent some hmacs from outside (for exchanges we know about)
    fn_table->C_DestroyObject=DestroyObjectOverride; // Otherwise people can just delete data and roll back to
    fn_table->C_SetAttributeValue=SetAttributeValueOverride; // Otherwise people can just overwrite data and r
    printf("Registering taurus FM with chunking support\n");
    initialized = false;
    return FMSW_RegisterDispatch(FM_NUMBER_CUSTOM_FM, handler);
}
```

2/6: Enforce secure configuration

Custom code can enforce that attributes of PKCS#11 objects are the most restrictive, and stop its operations otherwise

Ex: Ensure that secret key are marked as CKA_SENSITIVE and not CKA_EXTRACTABLE.

```
CK_ATTRIBUTE len_attrs[] = {
    {CKA_VALUE_LEN, &tmplen, s
    {CKA_SENSITIVE, &is_sensit
    {CKA_PRIVATE, &is_private,
    {CKA_MODIFIABLE, &is_modif
    {CKA_WRAP, &is_wrap, sizeo
    {CKA_EXPORT, &is_export, s
    {CKA_IMPORT, &is_import, s
    {CKA_UNWRAP, &is_unwrap, s
    {CKA_EXTRACTABLE, &is_extr
    {CKA_EXPORTABLE, &is_expor
    {CKA_DERIVE, &is_derive, s
    {CKA_ENCRYPT, &is_encrypt,
    {CKA_DECRYPT, &is_decrypt,
    {CKA_SIGN, &is_sign, sizeo
    {CKA_VERIFY, &is_verify, s
    {CKA_CLASS, &obj_class, si
};
```

3/6: In-HSM business logic

Move business logic from servers/VMs to the HSM

Ex: Create blockchain transactions (signature, payload) after enforcing a multi-sig quorum and governance rules

Benefits:

- Computation integrity and confidentiality protected
- Can interact with in-HSM crypto objects

Risks: Bugs leading to secrets leak or code execution

4/6: Application-level *AC

- Roles = **users** (request approvers), **admins** (rules approvers)
- Admins sign rules defining authorized quorums
- Users and admins sign with **hardware tokens**

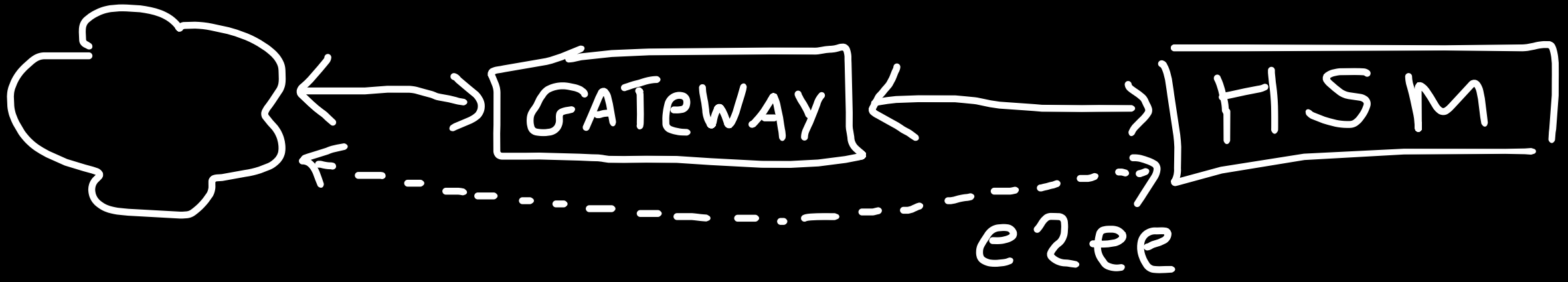
Only admin pubkeys in the HSM

Tricks needed to prevent replay and downgrade

5/6 Application-level secure channel

HSMs may support secure channels, but only at the network level, or offer insufficient security (anon DH in old HSMs)

If the consumer of the HSM response is not the host talking to the HSM, application-level security is needed (aka e2ee)



6/6: Minimize black-boxing

The proprietary HSM code is generally not open-source, therefore harder to review for bugs, let alone fix them

Alternative: integrate code from auditable/OSS libraries via the custom modules (may need tweaks/optimization/stripping)

Exception: **randomness**: HSM's PRNG and entropy sources

Can post-process with a custom DRBG

Why a state?

Stateless HSMs are convenient and simple to manage

- Multiple instances behind a load balancer
- Immutable state configured once in a key ceremony

However, **statefulness** often needed for

- Anti-replay, anti-downgrade (ex: monotonous counter)
- Enforcement of security policies (ex: via timestamps)

Challenges of HSM states

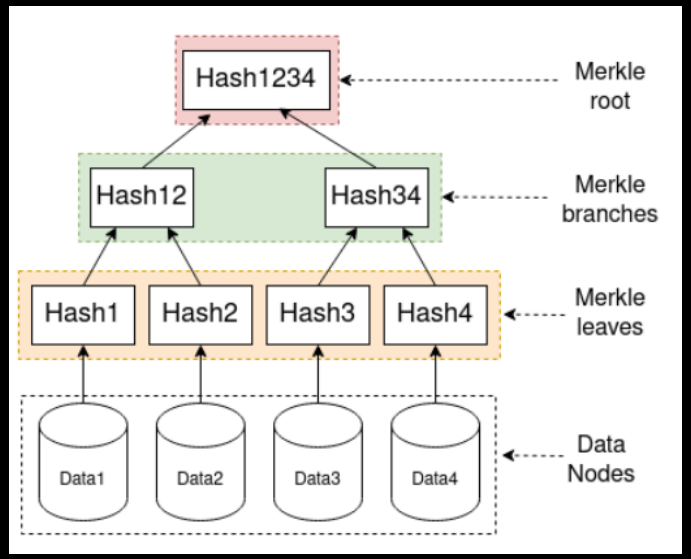
- HSMs' storage is limited, and I/O is slow
- High-availability needs at least 2 redundant HSMs
- State bounded in size (must fit in a ~2MB message)
- State transitions must be verifiable

Solution: **trees**!

Merkle trees & Merkle proofs

Principle: only reveal state components needed by a request

- Encode the state as a Merkle tree
- Admins sign the root, verified in the HSM
- Merkle proofs



What if the state (thus root) changes?

How to **verify state changes given a partial state**?

Merkle trees limitations

A root represents a list of data nodes

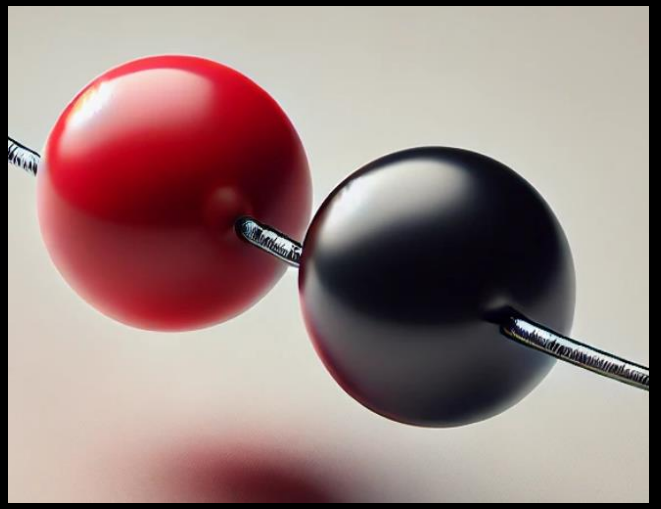
Logarithmic membership proof size

Read-only trees are easy…

How to insert/delete?

- Where to insert the data?
- How to efficiently “rebalance” the tree?

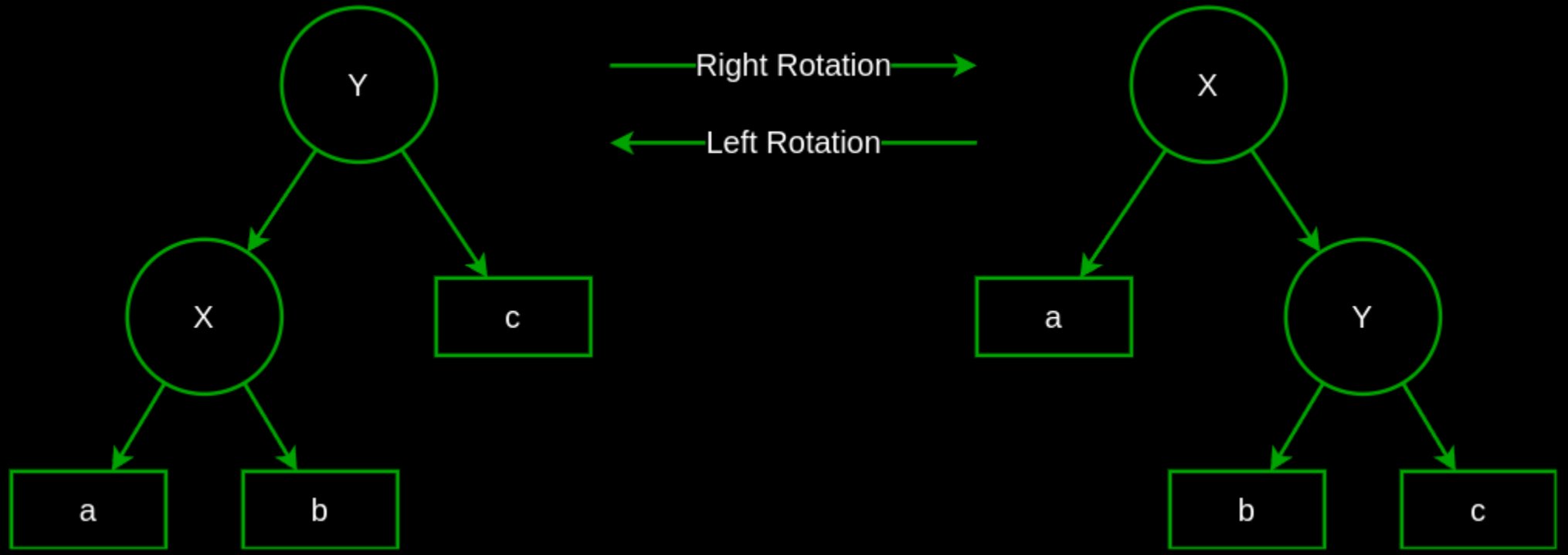
Red-black trees



- Allow updates on **partial** trees
- Keep Merkle-tree property
- Bounded height of at most 2log(*N*+1) with *N* nodes
- **Self-balancing** via simple “coloring rules”
  - RB1: Root is black
  - RB2: Any path from a node to a leaf has the same number of black nodes
  - RB3: There can’t be an edge between two red nodes

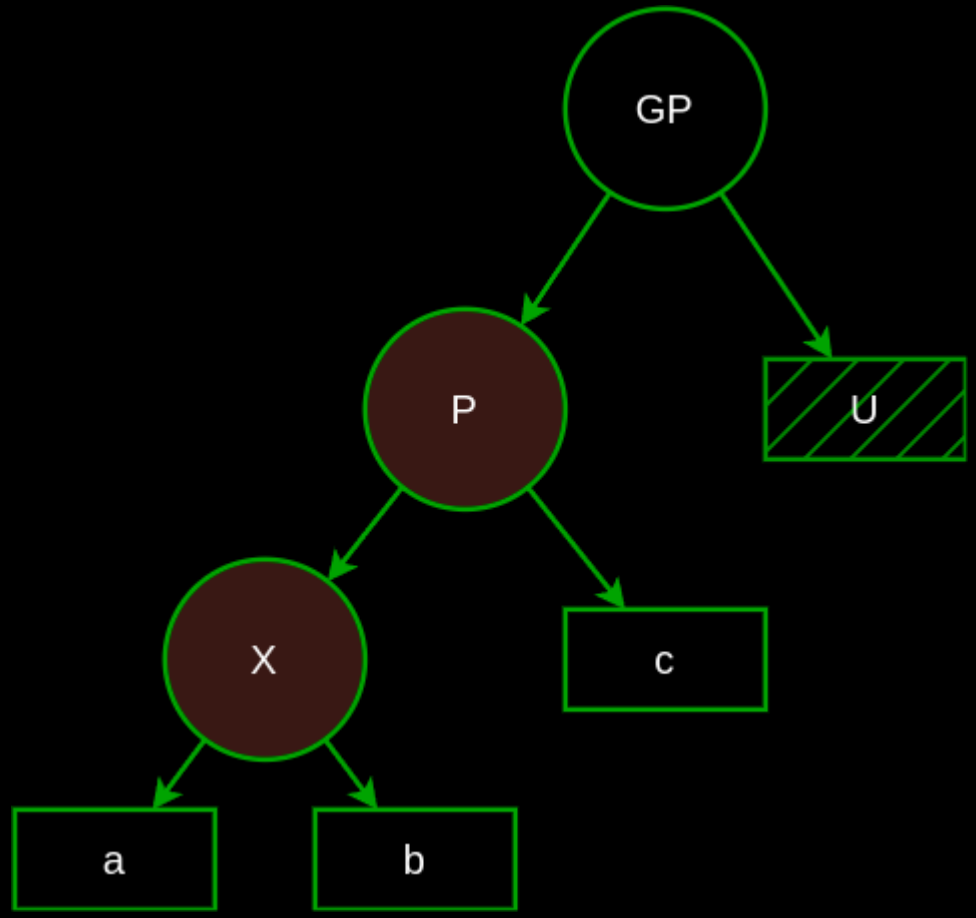
Tree examples

- Rebalancing performed through **rotations**
- Rotated subtrees preserve **RB** and **Merkle** properties



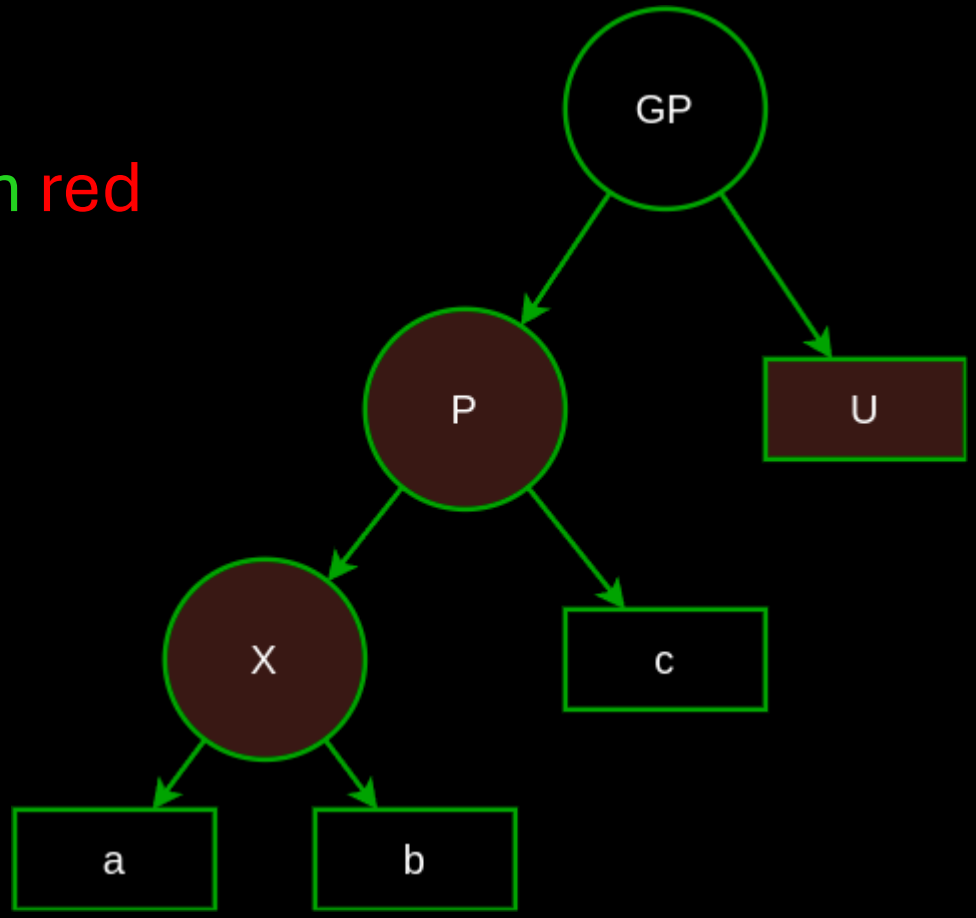
Tree insertion

- Rebalancing is **recursive** over the height of the tree
- Carries on so long as the parent P is **red**



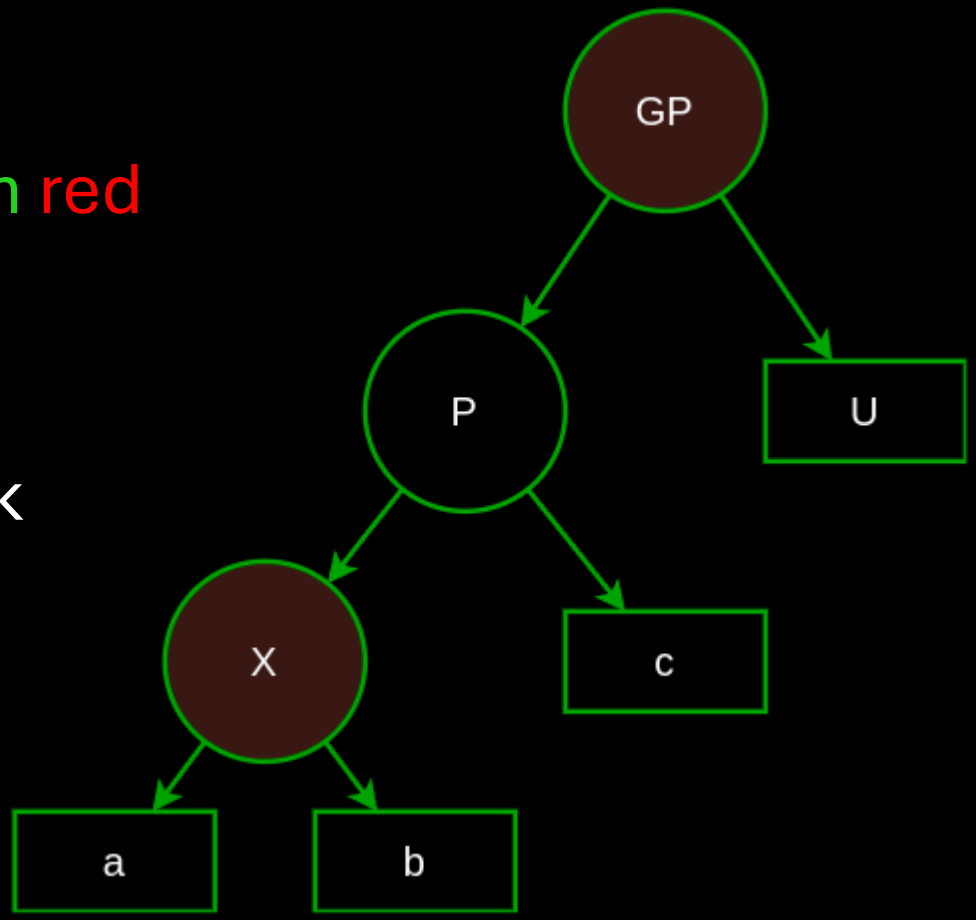
Tree insertion

- Case 1:
  - Parent **P** and uncle **U** are both red



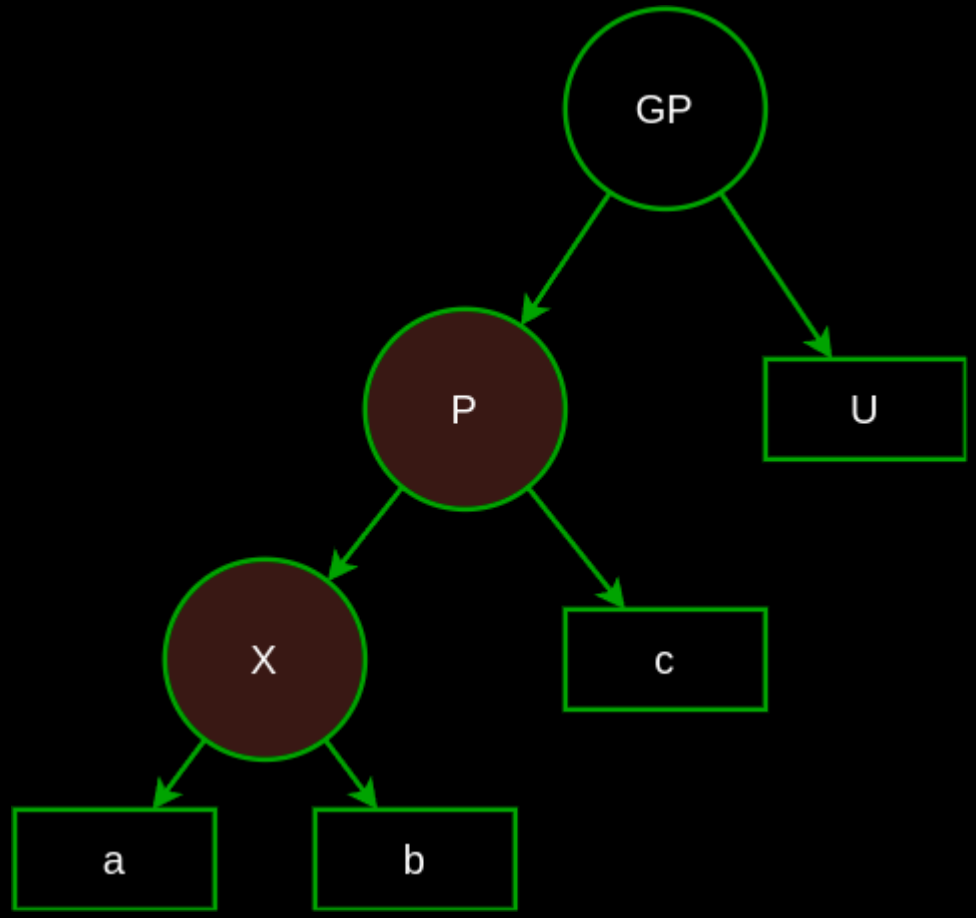
Tree insertion

- Case 1:
  - Parent **P** and uncle **U** are both red
- Solution:
  - Recolor both **P** and **U** to black
  - Recolor **GP** to red
  - No impact on subtrees



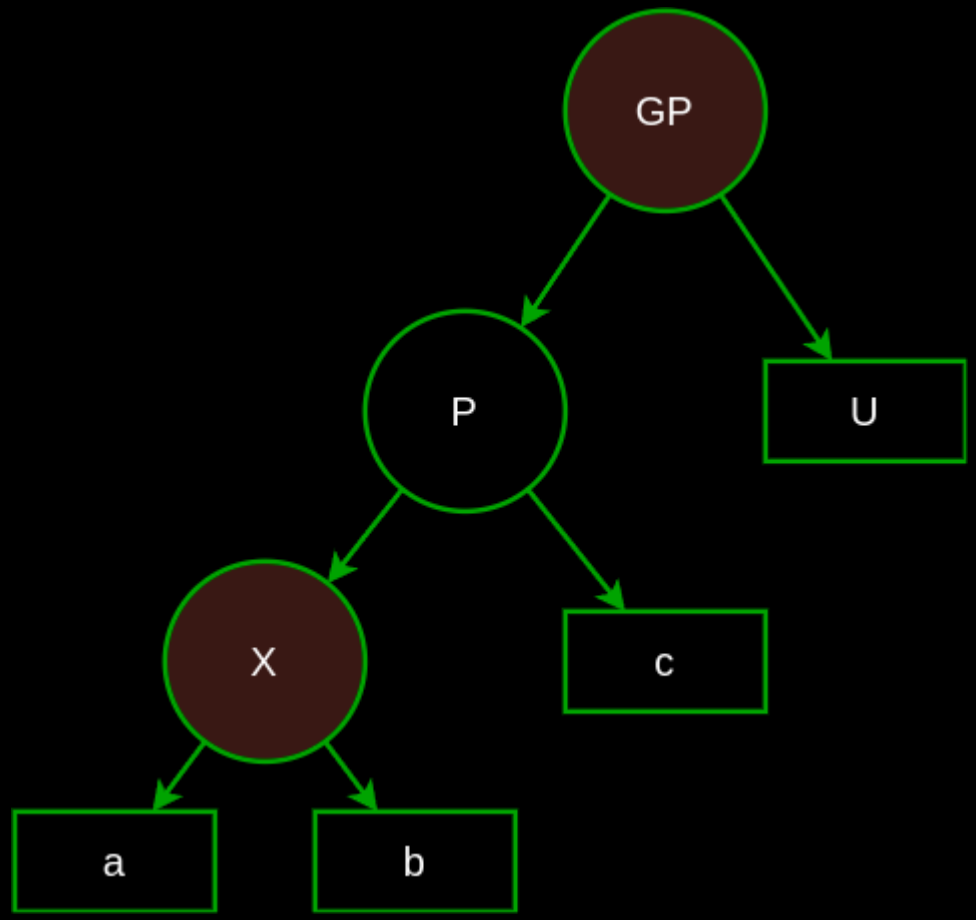
Tree insertion

- Case 2:
  - Uncle **U** is black
  - **X** is the left child of **P**



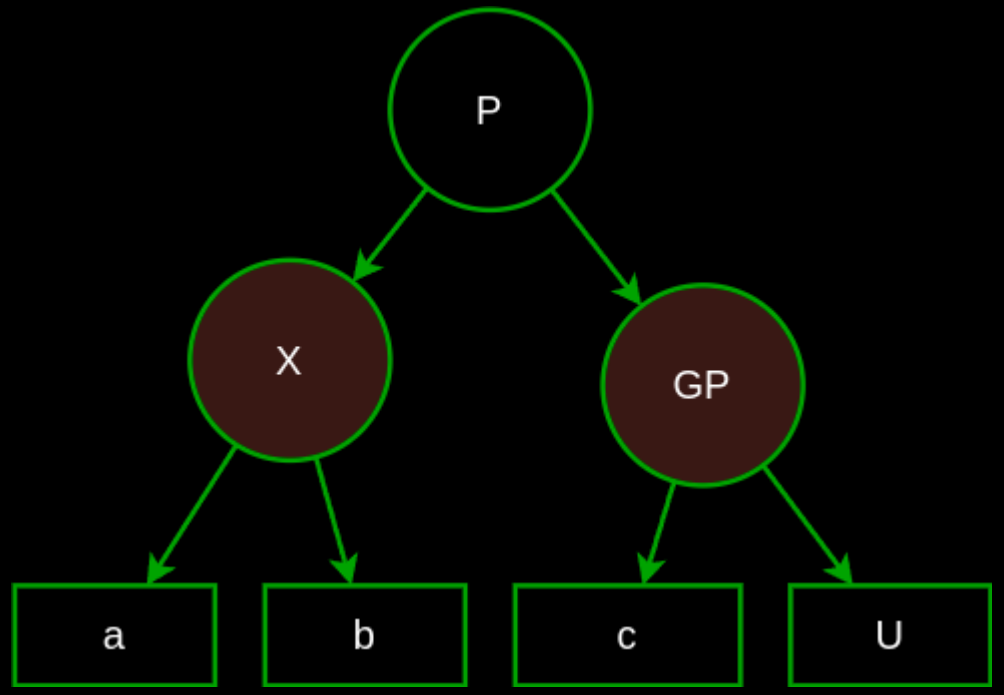
Tree insertion

- Case 2:
  - Uncle **U** is black
  - **X** is the left child of **P**
- Solution
  - Recolor **P** and **GP**
    - Breaks RB2



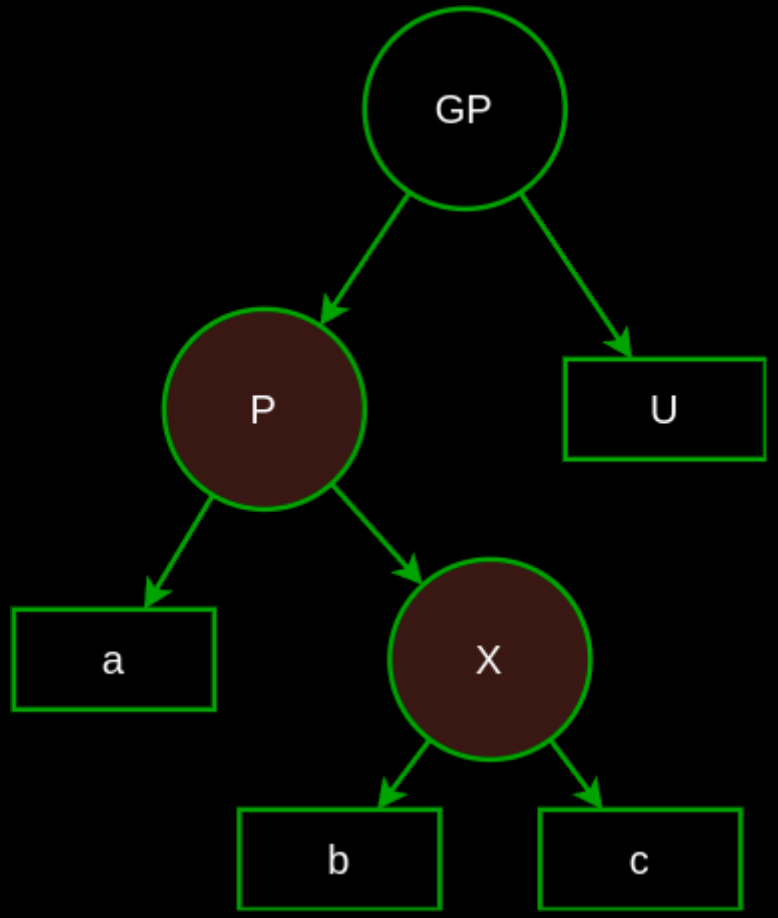
Tree insertion

- Case 2:
  - Uncle **U** is black
  - **X** is the left child of **P**
- Solution
  - Recolor **P** and **GP**
  - Rotate **GP** to the right
  - Subtrees not impacted



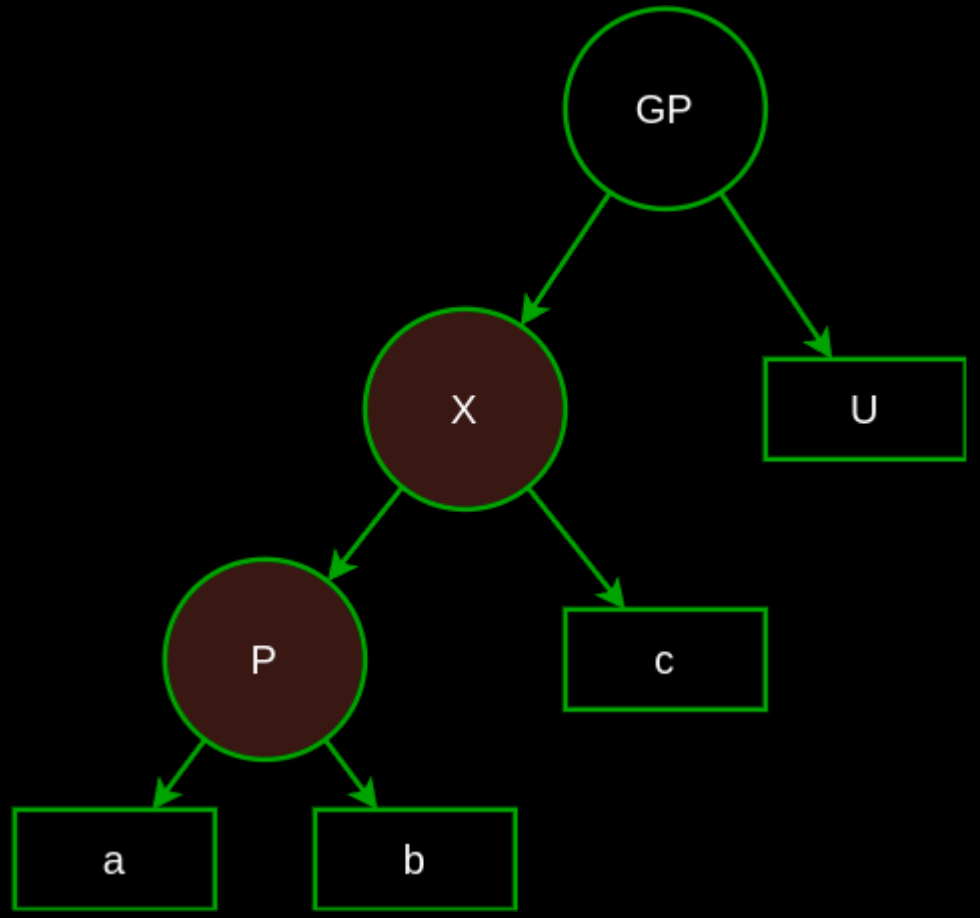
Tree insertion

- Case 3:
  - Uncle **U** is black
  - **X** is the right child of **P**



Tree insertion

- Case 3:
  - Uncle **U** is black
  - **X** is the right child of **P**
- Solution
  - Rotate **P** to the left
  - Brings us back to case 2 with **P** and **X** swapped



Tree conclusion

- **Red-black** and **Merkle** properties can be combined in a single structure
- Lets us perform **state transitions** on large datasets **within** a low-memory **HSM**
- Inserting a user to a set of 1M requires revealing 20-40 users

Best practices (1/2)

Software defense

- Keep the HSM firmware updated
- Tighten PKCS#11 attributes (to the minimum needed)
- Enable security features (secure channel)
- Custom code: minimize dependencies
- Custom code: have solid build/deploy integrity (see SLSA.dev)

Best practices (2/2)

Access control

- Segregate accesses and credentials (admin/SO, slot user/SO)
- Minimize network exposure (no internet facing, whitelisting)

Key management

- Generate critical keys in key ceremonies (in- or off-HSM?)
- Have reliable & tested back-ups and DR procedures

Use **HSM back-up/cloning**?

On cloud HSM aka HSMaaS

Convenient cloud-based systems, notably as KMS back-end

Limitations:

- Access may be indirect via some cloud middleware
- May be multi-tenant, sharing hardware with other users
- Limited capability to configure the HSM and PKCS#11 settings
- Impossible to run custom code
- How to be sure it’s really an HSM and not an emulator?

Conclusion

HSMs + in-HSM custom logic is a powerful setup suitable for various high-assurance security systems, but requires significant investment in

- Bespoke hardening to reduce the attack surface
- Management of compute and storage limitations
- SDLC integrity and QA
- HSM model/vendor-specific shenanigans


TAURUS
Thank you
A joint work with the Taurus team
Acknowledgements:
André S., Antony V., Mattia T., Ryan H., Stefano Z., Tal B.
https://taurushq.com