Digests as Secret Keys (DASK) for PQC solutions

Pau Homs Calafell

30 de juny de 2025

Resum– Aquest projecte desenvolupa una arquitectura híbrida per a la generació de claus privades resistents a atacs quàntics, pensada per integrar-se en l'ecosistema actual de criptomonedes. A partir de signatures basades en hash i la construcció d'un arbre de Merkle, es deriva una clau privada compatible amb sistemes de corba el·líptica com els utilitzats a Bitcoin. La clau pública corresponent i l'adreça Bitcoin generada permeten mantenir la compatibilitat amb l'arquitectura actual. A més, el sistema permet signar i verificar transaccions, emmagatzemar múltiples signatures i automatitzar tot el procés mitjançant *scripts* modulars. Aquesta proposta demostra que és viable reforçar la seguretat post-quàntica en criptomonedes sense modificar el protocol base.

Paraules clau– Criptografia post-quàntica, Signatures basades en hash, Arbres de Merkle, claus ECC, Bitcoin, Seguretat

Abstract– This project develops a hybrid architecture for generating quantum-resistant private keys, designed to integrate with the current cryptocurrency ecosystem. Using hash-based signatures and the construction of a Merkle tree, a private key is derived that remains compatible with elliptic curve systems like those used in Bitcoin. The corresponding public key and generated Bitcoin address ensure compatibility with the existing architecture. Furthermore, the system enables transaction signing and verification, storage of multiple signatures, and full automation of the process through modular scripts. This proposal demonstrates that it is feasible to strengthen post-quantum security in cryptocurrencies without altering the underlying protocol.

Keywords– Post-quantum cryptography, hash-based signatures, Merkle trees, ECC keys, Bitcoin, Security

1 Introducció

En els darrers anys, l'aparició dels ordinadors quàntics ha plantejat un repte significatiu per a la seguretat dels sistemes criptogràfics clàssics. Esquemes utilitzats [1], com RSA, DSA o ECC, sobre els quals es fonamenta gran part de la infraestructura digital actual, inclosos els sistemes de pagament, els certificats digitals i les criptomonedes, podrien veure's compromesos per l'eficiència dels algorismes quàntics com el de Shor [2], que permet factoritzar nombres enters molt grans de forma extremadament eficient.

En aquest context, la criptografia post-quàntica [4] (PQC, per les sigles en anglès) apareix com una branca de la criptografia que busca esquemes resistents als possibles atacs quàntics per tal d'oferir alternatives segures. Dins de les diverses propostes, les signatures basades en hash [5] (HBS) es consideren una de les millors opcions, ja que són de les més segures, simples i de resistència coneguda.

El meu treball proposa una arquitectura híbrida per a la generació de claus privades basada en esquemes HBS, adaptada a la blockchain de Bitcoin [6]. A partir de la combinació de signatures hash amb la construcció d'arbres de Merkle [7], s'hi planteja derivar una clau ECC, compatible amb les d'adreces de Bitcoin. Això permet integrar una capa resistent a atacs quàntics sense haver de modificar el protocol existent.

El projecte inclou la implementació de diversos esquemes de firma hash, el disseny d'un sistema modular per a la generació i validació de transaccions. Aquesta proposta busca contribuir a la implementació de mecanismes criptogràfics més robustos que permetin preparar les criptomonedes davant l'arribada d'un ordinador quàntic.

• E-mail de contacte: pauhoca@gmail.com
• Menció realitzada: Tecnologies de la Informació
• Treball tutoritzat per: Jordi Herrera Joancomarti (DEiC)
• Curs 2024/25

2 OBJECTIUS

L'objectiu principal d'aquest treball és implementar un sistema de generació de claus que combini HBS amb ECC, obtenint un model resistent a atacs quàntics i compatible amb la infraestructura actual de Bitcoin.

Els objectius específics són:

- Analitzar les vulnerabilitats que pot presentar l'aparició d'un ordinador quàntic.
- Estudiar diversos esquemes de signatures digitals post-quàntiques basades en hash, com Lamport OTS, Winternitz OTS+, Merkle Signature Scheme (MSS) i SPHINCS+.
- Construir un arbre de Merkle a partir de les claus públiques dels esquemes HBS i derivar, a partir de l'arrel, una clau privada compatible amb ECC.
- Generar una adreça funcional de Bitcoin *Pay to Witness Public Key Hash* [8] (P2WPKH) sobre la xarxa testnet [9] a partir de la clau pública corresponent, seguint els estàndards de la xarxa.

3 METODOLOGIA I PLANIFICACIÓ

El desenvolupament del projecte s'ha estructurat a partir de la metodologia SCRUM [10]. S'ha realitzat de manera autònoma amb reunions bisetmanals amb el tutor i s'han anat fent entregues periòdiques per a millorar el projecte en cada iteració. A continuació es detalla la planificació:

- **Investigació preliminar (2 setmanes)**
  - Recopilació de bibliografia i revisió d'articles científics (1 setmana).
  - Anàlisi dels riscos quàntics en criptomonedes (1 setmana).
- **Estudi dels esquemes HBS (4 setmanes)**
  - Estudi de Lamport OTS 4.1.1 (1 setmana).
  - Estudi de Winternitz OTS+ 4.1.2 (1 setmana).
  - Estudi de MSS 4.1.3 (1 setmana).
  - Estudi de Sphincs+ 4.1.4 (1 setmana).
- **Implementació (10 setmanes)**
  - Desenvolupar una prova de concepte en un entorn controlat i simulat (8 setmanes).
  - Testejar el rendiment i ajustaments (2 setmanes).
- **Avaluació dels resultats (2 setmanes)**
  - Anàlisi de la seguretat i rendiment de l'esquema implementat (2 setmanes).

El projecte s'ha organitzat en un repositori de GitHub en diverses carpetes. Cadascuna d'elles conté el programa per generar l'esquema corresponent, junt amb els fitxers de guardat que es generen. Hi ha una carpeta per cada esquema, una per l'arbre de Merkle i una per tota la generació de claus ECC i Bitcoin. Addicionalment, es crea una carpeta i un fitxer JSON quan es realitza una signatura d'una transacció.

4 ESTAT DE L'ART

L'arribada de la computació quàntica representa una amenaça directa als sistemes criptogràfics clàssics, especialment als basats en problemes matemàtics com la factorització o el logaritme discret. Aquest risc ve de l'algorisme de Shor [2], que aprofita la superposició i l'entrellaçament quàntic per resoldre aquests problemes matemàtics en temps polinòmic, mentre que els algorismes clàssics funcionen en temps exponencial. Això significa que claus que actualment es consideren segures podrien trencar-se eficientment un cop es disposi d'un ordinador quàntic amb prou qubits [3] i una taxa d'error baixa.

Els atacs quàntics també poden afectar aspectes del funcionament de les criptomonedes, com ara la revelació prematura de claus privades durant el procés de transmissió de la transacció per la xarxa. Per exemple, si un node maliciós amb capacitats quàntiques intercepta una clau pública i pot calcular-ne la privada abans que es validi el bloc, podria robar els fons abans que el propietari real els recuperi.

Aquesta amenaça ha impulsat el desenvolupament de la criptografia post-quàntica, un camp que, gràcies a iniciatives com el procés d'estandardització del NIST [11], ha anat guanyant forma i reconeixement. Dins d'aquest marc, una de les millors opcions són les signatures digitals basades en funcions hash (HBS), conegudes per oferir una seguretat robusta basada en propietats ben estudiades com la resistència a col·lisions i la dificultat d'inversió.

4.1 Esquemes de signatures basades en hash

Els esquemes de signatures digitals basades en hash es poden classificar segons si requereixen o no mantenir estat durant el procés de signatura. Aquesta classificació, representada també a la Figura 1, distingeix entre:

- **Esquemes amb estat (stateful):** sistemes com XMSS (Extended MSS) i MSS necessiten portar un registre de les claus que ja han estat utilitzades per garantir que cada clau privada només es fa servir una vegada. Si es perd aquest registre, la reutilització de claus pot comprometre la seguretat. Tot i aquesta dificultat, ofereixen una base sòlida, habitualment basada en arbres de Merkle. Dins aquest grup també s'hi troben esquemes com Lamport OTS i Winternitz OTS+, que serveixen com a blocs fonamentals dins MSS i altres construccions.
- **Esquemes sense estat (stateless):** com SPHINCS+, no requereixen portar cap registre de l'estat intern entre signatures. Això simplifica la seva integració en sistemes pràctics, ja que elimina el risc associat a la pèrdua o corrupció d'aquest estat. Això ho fa especialment útil per a sistemes reals on portar un control d'estat seria complicat o poc viable.

En aquest treball s'han estudiat i implementat quatre mecanismes:

- **Lamport OTS**, **Winternitz OTS+** i **Merkle Signature Scheme (MSS)** pertanyen a la categoria d'esquemes amb estat.
- **SPHINCS+** forma part dels esquemes sense estat i és, actualment, un dels més avançats i escollits com a estàndard pel NIST [11].

Aquests esquemes han estat escollits pel tutor, principalment per la seva aplicabilitat directa en contextos pràctics com l'ecosistema de les criptomonedes.

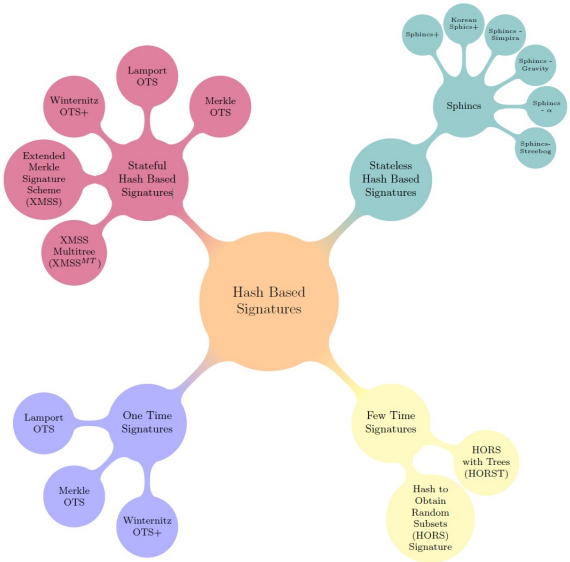

Fig. 1: Classificació dels esquemes de signatures basades en hash. Font: *An Overview of Hash-Based Signatures* [5].

A continuació, s'expliquen els mecanismes estudiats en aquest treball, que es poden consultar amb més detall al document [5]:

4.1.1 Lamport OTS

És considerat el punt de partida de les signatures digitals basades en hash. Lamport One-Time Signature genera claus d'un sol ús aplicant funcions hash a valors secrets aleatoris. La seva simplicitat el fa molt segur, però només serveix per a una única signatura. Per a un missatge de n bits, es generen $2n$ valors secrets aleatoris que formen la clau privada. La clau pública es forma aplicant una funció hash a cadascun d'aquests valors (veure Figura 8 de l'apèndix).

Per signar, se seleccionen els valors secrets corresponents als bits del missatge. La signatura es verifica comparant les imatges hash amb la clau pública. Com el nom indica, només pot ser utilitzada una vegada: si es reutilitza, es filtra part de la clau privada i es compromet la seguretat.

4.1.2 Winternitz OTS+

WOTS+ és una millora de Lamport que redueix la mida de la signatura i el cost computacional. Permet representar més bits per element de la clau privada mitjançant un sistema de base w. La base w en l'esquema Winternitz OTS+ és un paràmetre que serveix per agrupar els bits del missatge abans de signar-lo. En lloc de fer una correspondència 1 a 1 entre bits i valors secrets (com passa en Lamport OTS), Winternitz transforma blocs de bits (en base w) en valors enters. Això permet reutilitzar els mateixos valors secrets més vegades i, per tant, reduir la mida de la signatura i el nombre de claus necessàries. Per exemple, si el missatge té 256 bits i es treballa en base $w = 16$, es poden representar els 256 bits com 64 valors en base 16. Cada valor determina quantes vegades cal aplicar la funció hash a una part de la clau privada per construir la signatura.

A més, afegeix un checksum per protegir contra certes manipulacions. La verificació es basa en aplicar múltiples cops la funció hash segons el valor dels fragments del missatge. Aquest sistema ofereix una millor eficiència amb una lleugera pèrdua de simplicitat.

4.1.3 Merkle Signature Scheme (MSS)

Lamport OTS i WOTS+ tenen la limitació que només poden signar un únic missatge amb cada clau. Per poder reutilitzar la infraestructura i permetre vàries signatures, s'han d'encapsular molts esquemes OTS dins d'una estructura més gran. Per resoldre aquest problema, el *Merkle Signature Scheme* ens és molt útil.

El MSS fa ús d'un arbre de Merkle, una estructura de dades en forma d'arbre binari utilitzada habitualment en criptografia per agrupar múltiples valors de forma eficient. Cada fulla de l'arbre representa una clau pública d'un esquema OTS (com Lamport o WOTS+). A mesura que s'agrupen de dues en dues, es calculen els seus hashos combinats fins a arribar a un únic valor final: l'arrel de l'arbre. Aquest valor actua com a clau pública global del sistema.

Quan es vol signar un missatge, s'utilitza una de les claus OTS disponibles i es proporciona, juntament amb la signatura, la ruta d'autenticació dins l'arbre, anomenats complementaris. Aquesta ruta permet verificar que la clau OTS emprada forma part de l'arrel de l'arbre sense haver de mostrar la resta de claus.

D'aquesta manera, MSS permet signar múltiples missatges mantenint la base de seguretat dels esquemes OTS, a la vegada que es millora l'escalabilitat i es redueix la mida de la clau pública general. Aquesta combinació ha estat clau en el desenvolupament d'esquemes més avançats com XMSS o SPHINCS+.

A la Figura 2 tenim representat de forma visual un arbre binari de Merkle.

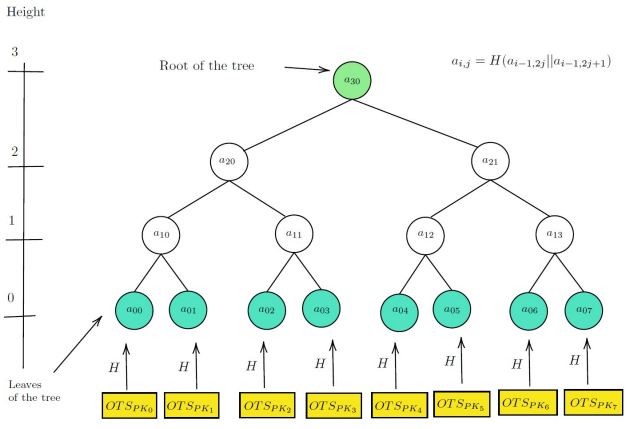

Fig. 2: Visualització del funcionament d'un arbre de Merkle. Font: *An Overview of Hash-Based Signatures* [5].

4.1.4 SPHINCS+

SPHINCS+ és un esquema sense estat i estandarditzat pel NIST. Es basa en l'ús d'una jerarquia d'arbres de Merkle (anomenats *hypertree*) que encapsulen esquemes WOTS+ i FORS. Un dels avantatges més destacats és que no cal mantenir estat entre signatures, a diferència de MSS o XMSS, tal com s'explica a la Secció 4.1. Això permet evitar la gestió complexa de claus ja utilitzades i millora la robustesa del sistema davant errors o pèrdua d'informació.

A més, s'ofereix la possibilitat d'ajustar la seguretat a través de diferents paràmetres: la profunditat de l'arbre, la mida de les claus i de les signatures, o les configuracions pròpies de WOTS+ i FORS. Aquesta flexibilitat permet adaptar l'esquema a diferents situacions buscant un equilibri entre seguretat i eficiència. Aquesta versatilitat l'ha fet ideal per a la integració en sistemes moderns post-quàntics. Es pot veure un esquema general de la construcció de SPHINCS+ a la Figura 9.

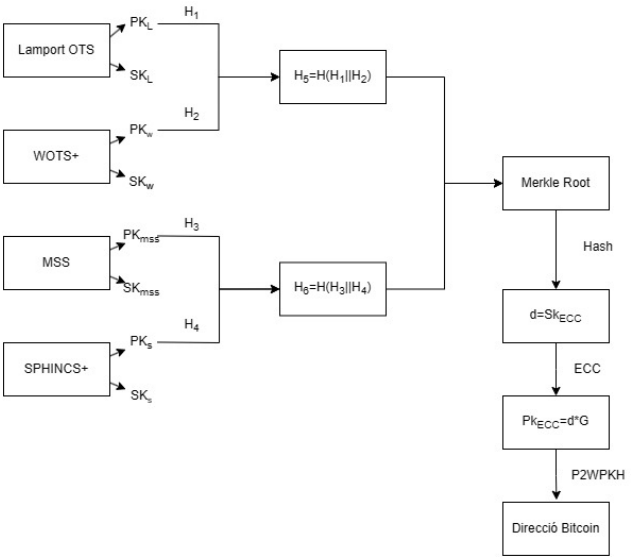

Fig. 3: Diagrama de l'arquitectura del sistema. Font: Pròpia.

5 DISSENY I IMPLEMENTACIÓ DEL SISTEMA

Aquest apartat descriu l'arquitectura i el procés tècnic que permet la generació d'una clau privada resistent a atacs quàntics, derivada d'un arbre de Merkle que implementa quatre esquemes HBS i la seva integració amb el sistema de claus públiques d'ECC utilitzat a Bitcoin. L'objectiu és construir un model que ofereixi resistència post-quàntica i mantenir-hi la compatibilitat.

5.1 Arquitectura híbrida de generació de claus

L'arquitectura proposada combina els esquemes HBS amb les claus de corba el·líptica utilitzades a Bitcoin. La idea principal és utilitzar les claus generades per cada HBS com a fulles d'un arbre de Merkle, del qual s'obtindrà una arrel. A partir de l'arrel, es deriva una clau privada aplicant-li SHA-256 [12], que es farà servir per generar una clau pública ECC vàlida. Aquesta clau pública s'utilitza per crear una adreça Bitcoin de tipus P2WPKH per a la testnet. A la Figura 3 podem veure representat esquemàticament aquesta arquitectura.

5.2 Generació dels esquemes HBS i construcció de l'arbre de Merkle

El procés comença amb la generació dels parells de claus per a cada esquema de signatures. Per als esquemes Lamport OTS, Winternitz OTS+ i MSS, es va seguir el document *An Overview of Hash-Based Signatures* [5] com a base teòrica, el qual explica amb detall el funcionament i el procés criptogràfic de cada esquema. A partir d'aquestes explicacions, es van desenvolupar scripts propis (anomenats `keygen_X.py`, on "X" és el nom de l'esquema) que generen les claus i les guarda en fitxers JSON separats. En cada fitxer de clau pública s'inclou també el seu hash, per facilitar el posterior càlcul de l'arbre de Merkle.

Per a aquests esquemes només es van utilitzar llibreries estàndard de Python com `hashlib`, `secrets`, `json`, `math` i `os`, i així no es depenia de llibreries externes ni de processos de compilació complexos. Això permet que es puguin desenvolupar i executar en qualsevol entorn compatible amb Python, tant a Windows com a Linux.

En el cas concret del MSS, es va implementar utilitzant Lamport OTS com a esquema de firma intern per a les fulles de l'arbre i es va obtenir, així, una estructura del tipus `MSS_LOTS`.

Pel que fa a SPHINCS+, es va enfocar d'una altra manera. Donada la seva complexitat i el fet que ja sigui un esquema estandarditzat, es va optar per utilitzar una llibreria ja existent: `pyspx` [13]. Aquesta llibreria proporciona *bindings* [14] en Python sobre la implementació oficial escrita en C, i permet treballar amb diversos paràmetres de SPHINCS+ de forma senzilla.

Tot i això, `pyspx` no és compatible amb sistemes Windows, ja que depèn d'eines de compilació pròpies d'Unix com `make` i `gcc`. Per tal de fer-la funcionar correctament, es va preparar una màquina virtual amb Ubuntu i es va compilar la llibreria en aquest entorn. Un cop configurat, el seu ús es va integrar al sistema de forma modular, sense afectar l'execució de la resta d'esquemes ni la compatibilitat global del projecte.

Un cop s'han generat tots els parells de claus, es crea l'arbre. Primer de tot, s'importen els hashos prèviament calculats de les claus públiques de cada esquema com a fulles de l'arbre i es calcula l'arbre de Merkle donant com a resultat la seva arrel. Un cop creada, es guarda en un JSON (veure Figura 4 de l'apèndix) l'arrel en hexadecimal, els hashos de cada fulla i els seus complementaris. Els complementaris ens serviran per si posteriorment volem verificar una firma, ja els tinguem guardats i no sigui necessari recalcular-los.

5.3 Derivació de claus ECC a partir de l'arrel Merkle

Obtinguda l'arrel de l'arbre de Merkle, aquesta es fa servir per generar una clau privada vàlida per a Bitcoin. Aquesta arrel, que conté la informació condensada de totes les claus

públiques dels esquemes HBS, actua com a entrada a la funció `generate_private_key_from_merkle_root()` dins de l'script `ecc_keys.py`.

Primerament, es calcula el hash SHA-256 sobre l'arrel per obtenir un valor uniforme de 32 bytes que actuarà com a clau secreta dins la corba el·líptica SECP256K1 [15].

Tot seguit, per fer-la compatible amb el format estàndard de Bitcoin, s'hi afegeix un prefix identificatiu de la xarxa: `0xEF` per a la testnet (o `0x80` en el cas de la mainnet) i un checksum de 4 bytes obtingut amb un doble SHA-256 amb l'objectiu de controlar possibles errors. El resultat es codifica en format **Wallet Import Format (WIF)** [16] mitjançant l'esquema `Base58Check`, que genera una cadena alfanumèrica fàcil d'utilitzar i reconèixer.

Aquesta clau privada codificada s'instancia com un objecte `PrivateKey` que ens proporciona la llibreria `bitcoinutils` [17]. Aquesta instància permet operar directament sobre la clau, derivar-ne la clau pública, o signar transaccions sobre la corba SECP256K1. El valor de la clau privada es guarda al fitxer `ecc/ecc_private_key_wif.txt`.

Tot i que també es guarda la clau pública derivada al fitxer `ecc/ecc_public_key_hex.txt`, aquest pas no és estrictament necessari i no dona cap benefici pràctic, ja que sempre es pot tornar a obtenir a partir de la clau privada mitjançant el mètode `.get_public_key()`. S'ha decidit mantenir aquest pas per mantenir la traçabilitat i per finalitats de verificació durant el desenvolupament.

El procés de derivació i representació d'aquestes claus es pot veure a la Figura 10, mentre que el de guardat es mostra a la Figura 11 de l'apèndix.

5.4 Creació d'adreça Bitcoin testnet i integració amb ECC

Amb la clau privada generada, el següent pas consisteix en generar l'adreça Bitcoin corresponent. Aquest procés completa la integració entre l'arrel Merkle i Bitcoin basat en ECC.

A partir de la clau privada, es deriva la pública utilitzant el mètode `.get_public_key()` de la classe `PrivateKey`.

Per generar una adreça vàlida dins la testnet de Bitcoin, s'utilitza el mètode `.get_segwit_address()`, que transforma la **clau pública** en una adreça en format P2WPKH, codificada segons l'estàndard Bech32 [8]. Aquest tipus d'adreça comença amb el prefix `tb1`, que identifica clarament que es tracta de la testnet.

El mètode `get_segwit_address()` encapsua diversos passos interns:

- El càlcul dels hashos necessaris (SHA-256 i RIPEMD-160) a partir de la clau pública.
- La construcció automàtica de l'script witness.
- La codificació en format Bech32 i la generació del checksum final.

L'entorn de treball es configura a l'inici del script amb la instrucció `setup('testnet')` per assegurar que totes les claus i adreces generades siguin vàlides dins de la xarxa de proves i no afectin la mainnet de Bitcoin.

Tot i que aquest projecte no implementa manualment la codificació el format Bech32, és interessant saber i entendre les seves propietats per justificar-ne l'ús:

- Fa ús d'un **prefix**, `tb1` per testnet o `bc1` per mainnet, i així facilita la seva identificació.
- Té un **format eficient**; és més compacte i tolerant a errors de transcripció.
- Redueix l'**ambigüitat visual**: Bech32 només utilitza caràcters minúsculs i evita dígits i lletres que es poden confondre amb facilitat, (0/O, 1/l), cosa que minimitza errors humans en la transmissió o introducció manual de l'adreça.

Aquest tipus d'adreces modernes són molt utilitzades en aplicacions reals, ja que redueixen les comissions associades a les transaccions i milloren la seguretat operativa.

Per exemple, a partir de la clau pública en hexadecimal:
`02f726e667d163d770dcd5aa73dd7a2532e9223fbb5594cc0acb79fa0b2fe0404f`

es va generar l'adreça:
`tb1qf3y5unhg3mfthyn6y0l3n9g99ty2evaj0y7gz8`

Aquestes dades es poden verificar amb eines externes com el *Bech32 address decoder* o utilitats oficials de verificació. Tot aquest procés està automatitzat dins l'script `ecc_keys.py`, i es pot veure esquemàticament a la Figura 14 de l'apèndix.

5.5 Scripts i eina pqc_generator.py

Tot el sistema ha estat encapsulat en scripts de Python que permeten una execució modular i automatitzada. S'ha creat l'script `pqc_generator.py`, que centralitza les crides als diferents mòduls del projecte i permet reproduir el flux complet de generació de claus i adreça. Aquest script segueix les etapes que es mostren a la captura que es pot consultar a la Figura 12 de l'Apèndix:

- Generació de claus per a cada esquema HBS.
- Construcció de l'arbre de Merkle i obtenció de l'arrel.
- Derivació de la clau privada ECC i guardat en format WIF.
- Generació de la clau pública i l'adreça Bitcoin.

D'aquesta manera, es facilita l'execució del sistema complet amb una única instrucció, mantenint la flexibilitat necessària per a modificar o ampliar qualsevol etapa. A més, fent-ho de manera modular, permet afegir nous esquemes de signatura o alterar la lògica de derivació de claus sense afectar la resta del projecte.

L'output de la seva execució a la Figura 13 de l'Apèndix.

6 RESULTATS

Aquest apartat recull els resultats obtinguts després d'haver implementat el sistema complet. Es presenta una validació funcional, exemples pràctics i una breu anàlisi del funcionament general de la proposta híbrida.

6.1 Validació del sistema: generació, signatura i verificació

Per comprovar que totes les parts del flux funcionen correctament, s'han realitzat tres tipus de proves:

1. **Execució completa del generador de claus.** L'script `pqc_generator.py` crida seqüencialment:
   - la generació dels parells de claus HBS i càlcul dels seus hashos,
   - la construcció de l'arbre de Merkle i obtenció de l'arrel,
   - la derivació de la clau privada ECC (aplicant SHA-256 sobre l'arrel) i codificació en WIF,
   - la derivació de la clau pública i la generació de l'adreça Bech32 P2WPKH de testnet.

   Després de diverses execucions, s'ha comprovat que tots els fitxers de sortida (`.json`, `.txt`) existien i contenien els valors esperats.

2. **Signatura de l'identificador de transacció.** Utilitzant `sign_tx.py`, s'ha validat la càrrega i l'ús de la clau privada WIF per signar un `tx_id` de prova:
   - `load_private_key()` llegeix el WIF i crea l'objecte `PrivateKey`.
   - `load_tx_id()` llegeix l'identificador de transacció i el converteix a hexadecimal.
   - `sign_tx_id(tx_id, priv)` utilitza `SigningKey.from_string()` de la llibreria ecdsa i signa determinísticament amb SHA-256, retornant la signatura en hexadecimal.
   - `save_signature()` guarda en `signatures/tx_sig.json` un objecte JSON amb camps `tx_id`, `signature` i `public_key`, afegint-lo sense sobreescriure entrades anteriors (veure Figura 6).

3. **Verificació de la signatura.** Amb `verify_tx.py` s'ha comprovat la validesa de la signatura:
   - Es llegeix el JSON `signatures/tx_sig.json` i es busca l'entrada que coincideix amb el `tx_id` de prova.
   - `verify_signature()` construeix un `VerifyingKey` a partir de la clau pública (no comprimida), reconverteix el missatge i la signatura de hexadecimal a bytes, i crida `vk.verify()`.
   - Es capturen errors de `BadSignatureError` i altres excepcions, mostrant un missatge corresponent i retornant `True` només si la verificació és correcta (veure Figura 7).

En totes les proves s'han obtingut resultats coherents:

- Les claus i l'arbre de Merkle es generen sense errors i coincideixen amb els hashos esperats.
- La signatura generada per `sign_tx.py` passa la verificació de `verify_tx.py` sense cap error.
- El format JSON de sortida és consistent, permetent afegir múltiples signatures i verificar-les de forma automàtica.

Això confirma que el flux complet és funcional i estable dins de l'arquitectura proposada.

6.2 Exemples pràctics i resultats obtinguts

Per il·lustrar el funcionament del sistema, es mostren els resultats d'una execució completa, que inclou la generació de claus, l'arbre de Merkle, la derivació de claus ECC i la generació de l'adreça Bitcoin, així com la signatura i verificació. Es pot veure l'output d'una execució completa a la Figura 13.

- **Claus públiques HBS:** s'emmagatzemen en format JSON, juntament amb el seu hash per facilitar el càlcul de l'arbre de Merkle.
- **Arbre de Merkle:** s'ha creat correctament a partir de les fulles (hashos de claus públiques) i es desa l'arrel, així com els hashos complementaris (veure Figura 4).

Fig. 4: Representació de l'estructura de guardat de l'arbre de Merkle. Font: Pròpia.

- **Claus ECC i adreça Bitcoin:** la clau privada derivada s'ha codificat en WIF, i la clau pública generada s'ha utilitzat per obtenir una adreça P2WPKH de testnet (veure Figura 5).

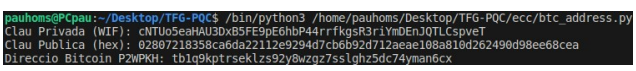

Fig. 5: Resultat de l'execució del script btc_address.py. Font: Pròpia.

- **Signatura d'una transacció:** s'ha signat un identificador de transacció (simulat) i s'ha desat el resultat en format JSON (veure Figura 6).

```
"signatures": [
    {
        "tx_id": "74623171396b70747273656b6c7a7339327938777a677a3773736c67687a3564633734796d616e366378",
        "signature":
        "67b7daf78a238cfe769c04f97fb52ec8253faded540db53dddcdc4170e364db815c8ac813368b79b1f988a1870f563aaf0e68365ef221e7d6cb664ad092
        91b53",
        "public_key":
        "04807218358ca6da22112e9294d7cb6b92d712aeae108a810d262490d98ee68cea09eb1810667989c4d58df53fe20f5dae7fd9f806bd446346a870a830e
        8abe2c6"
    }
]
```


Fig. 6: Fitxer de guardat de signatures. Font: Pròpia.

- **Verificació:** s'ha validat que la signatura generada és vàlida mitjançant els scripts propis del sistema (veure Figura 7).

```
pauhoms@PCpau:~/Desktop/TFG-PQC$ /bin/python3 /
home/pauhoms/Desktop/TFG-PQC/ecc/verify_tx.py
La signatura per la transaccio 74623171396b7074
7273656b6c7a7339327938777a677a3773736c67687a356
4633734796d616e366378 es valida.
```

Fig. 7: Verificació de la signatura d'una transacció. Font: Pròpia.

6.3 Discussió sobre l'eficiència i limitacions

El sistema proposat ha demostrat ser funcional i modular, però presenta algunes limitacions que cal tenir en compte de cara a la seva aplicació pràctica. Tot seguit es comenten els punts més rellevants en termes d'eficiència, portabilitat i escalabilitat:

- **Rendiment i càrrega computacional:** els esquemes de signatures basades en hash, especialment SPHINCS+, tenen una càrrega computacional alta en comparació amb esquemes clàssics com ECDSA. La generació de claus i signatures pot resultar lenta en entorns amb recursos més limitats.
- **Ocupació d'espai:** tant les claus públiques com les signatures d'alguns esquemes (com Lamport o Winternitz) ocupen més espai que les alternatives tradicionals. Això podria suposar un inconvenient en sistemes on l'emmagatzematge o l'amplada de banda siguin crítics.
- **Compatibilitat d'entorn:** la llibreria `pyspx` utilitzada per SPHINCS+ no és compatible amb Windows, perquè depèn d'eines de compilació pròpies d'Unix. Aquest fet obliga a utilitzar màquines virtuals o entorns Linux per poder executar el sistema complet.
- **Aplicabilitat limitada a testnet:** el sistema ha estat validat per la xarxa testnet de Bitcoin. La seva integració en entorns reals necessitaria més proves, incloent-hi seguretat en la gestió de claus i compatibilitat amb carteres i nodes reals.
- **No reutilització de claus:** tot i que esquemes com SPHINCS+ permeten múltiples signatures, d'altres com Lamport OTS estan pensats per signar una sola vegada. Aquest factor limita l'ús directe d'alguns esquemes en aplicacions habituals si no s'hi afegeixen capes com MSS.
- **Automatització i modularitat:** en positiu, l'estructura modular del projecte i l'automatització mitjançant `pqc_generator.py` faciliten l'ampliació futura del sistema. A més, es pot adaptar fàcilment per incorporar nous esquemes de firma o algoritmes de derivació de claus.

Tot i aquestes limitacions, el projecte demostra que és viable construir un sistema híbrid capaç de combinar la resistència post-quàntica amb la compatibilitat amb la infraestructura actual de Bitcoin, donant pas a futurs desenvolupaments més robustos i aplicables.

7 Conclusions

Aquest projecte ha proposat i implementat una arquitectura híbrida per a la generació de claus privades resistents a atacs quàntics, que es pot integrar dins l'ecosistema de Bitcoin sense modificar-ne el protocol base. A través de la combinació d'esquemes HBS, la construcció d'un arbre de Merkle i la derivació d'una clau privada compatible amb ECC, s'ha demostrat que és possible crear un sistema funcional, modular i compatible amb la infraestructura actual.

Tots els objectius definits al començament s'han assolit: he pogut estudiar i implementar diversos esquemes HBS (Lamport OTS, Winternitz OTS+, MSS i SPHINCS+), construir un arbre de Merkle a partir de les claus públiques i generar una adreça funcional de Bitcoin per a la testnet. Tot això, integrat en scripts modulars que automatitzen tant la generació com la signatura i la verificació.

El sistema és funcional i modular, però també ha tingut alguns punts crítics. Per exemple, SPHINCS+ només s'ha pogut executar en entorns Unix, i esquemes com Lamport presenten limitacions d'ús únic. Tot i això, el disseny modular ha permès fer proves i integracions de forma àgil, i obre la porta a futures ampliacions.

Més enllà de la implementació, aquest projecte vol posar en valor la necessitat de preparar els sistemes actuals per a un futur amb ordinadors quàntics. Algorismes com el de Shor poden posar en risc la seguretat d'esquemes com RSA o ECC, àmpliament utilitzats en criptomonedes. En aquest context, els esquemes basats en hash ofereixen una alternativa robusta: es fonamenten en funcions hash i arbres de Merkle, que actualment es consideren resistents a la computació quàntica. Això protegeix un dels punts més crítics del sistema com és el procés de generació de claus.

En un futur, es podria continuar ampliant el sistema amb nous esquemes post-quàntics, explorar-ne l'aplicació en altres *blockchains* (com Ethereum o Litecoin), o integrar-lo amb carteres reals per provar-ne l'ús en entorns més propers a la producció.

Personalment, aquest projecte ha estat molt més que una pràctica tècnica. M'ha permès connectar teoria i aplicació real, aprofundir en un tema que m'apassiona i aportar un petit gra de sorra a la idea sobre com fer més segures les criptomonedes davant un futur que cada cop és més proper.

Agraïments

Vull expressar el meu agraïment al meu tutor, Jordi Herrera Joancomarti, per haver-me donat l'oportunitat de desenvolupar un treball en l'àmbit de la criptografia aplicada a les criptomonedes, així com per la seva orientació i els seus consells al llarg de tot el projecte. També per haver estat

un dels professors que em va despertar la curiositat per la criptografia i la ciberseguretat.

Gràcies també als meus pares, que m'han acompanyat en tot moment i m'han ajudat quan ho necessitava.

I gràcies a la meva parella que, com a editora, m'ha ajudat amb la correcció i revisió de l'article.

REFERÈNCIES

[1] Amazon Web Services, "¿Qué es la criptografía?," AWS, 2025. [En línia]. Disponible a: https://aws.amazon.com/es/what-is/cryptography/

[2] Classiq, "Quantum Cryptography - Shor's Algorithm Explained," 19 jul. 2022. [En línia]. Disponible a: https://www.classiq.io/insights/shors-algorithm-explained

[3] IBM, "What is a Qubit?," *IBM Think*, 28 feb. 2024. [En línia]. Disponible a: https://www.ibm.com/think/topics/qubit

[4] NIST, "Post-Quantum Cryptography," Computer Security Resource Center, 2025. [En línia]. Disponible a: https://csrc.nist.gov/projects/post-quantum-cryptography

[5] K. Das, "An Overview of Hash-Based Signatures," Cryptology ePrint Archive, Report 2023/411, 2023. [En línia]. Disponible a: https://eprint.iacr.org/2023/411.pdf

[6] Autor desconegut, "Hash-Based Signature Schemes for Post-Quantum Bitcoin," Conduition, 21 oct. 2024. [En línia]. Disponible a: https://conduition.io/cryptography/quantum-hbs/

[7] Investopedia, "Merkle Tree," Investopedia, 26 jul. 2024. [En línia]. Disponible a: https://www.investopedia.com/terms/m/merkle-tree.asp

[8] Bitcoin Design Community, "Address," Bitcoin Design Guide, 2025. [En línia]. Disponible a: https://bitcoin.design/guide/glossary/address/

[9] Bitcoin Wiki, "Testnet," Bitcoin Wiki, 2024. [En línia]. Disponible a: https://en.bitcoin.it/wiki/Testnet

[10] Atlassian, "Qué es scrum y cómo empezar," Atlassian, 2024. [En línia]. Disponible a: https://www.atlassian.com/es/agile/scrum

[11] NIST, "NIST Announces First Four Quantum-Resistant Cryptographic Algorithms," National Institute of Standards and Technology, 5 jul. 2022. [En línia]. Disponible a: https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms

[12] Simplilearn, "What is SHA-256 Algorithm in Cyber Security?," Simplilearn, 2024. [En línia]. Disponible a: https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm

[13] sphincs, "pyspx: SPHINCS+ bindings and utilities for Python," GitHub, 2023. [En línia]. Disponible a: https://github.com/sphincs/pyspx

[14] K. Rajendran, "A Comprehensive Walkthrough of Python C Binding Creation," Medium, 29 aug. 2023. [En línia]. Disponible a: https://medium.com/@kapilanr2003/a-comprehensive-walkthrough-of-python-c-binding-creation-253a6d11c1ff

[15] Nervos, "secp256k1: un algoritmo clave," Nervos Knowledge Base, 25 aug. 2024. [En línia]. Disponible a: https://www.nervos.org/es/knowledge-base/secp256k1_a_key%20algorithm_(explainCKBot)

[16] Learn Me A Bitcoin, "WIF – Wallet Import Format," Learn Me A Bitcoin, 2024. [En línia]. Disponible a: https://learnmeabitcoin.com/technical/keys/private-key/wif/

[17] K. Karasavvas, "BitcoinUtilities.pdf," GitHub, 2025. [En línia]. Disponible a: https://github.com/karask/python-bitcoin-utils/blob/master/BitcoinUtilities.pdf

[18] Investopedia, "SegWit (Segregated Witness)," Investopedia, 2024. [En línia]. Disponible a: https://www.investopedia.com/terms/s/segwit-segregated-witness.asp

[19] Cointelegraph, "¿Qué es la Lightning Network de Bitcoin y cómo funciona?," Cointelegraph en Español, 13 feb. 2024. [En línia]. Disponible a: https://es.cointelegraph.com/learn/articles/what-is-the-lightning-network-in-bitcoin-and-how-does-it-work

APÈNDIX

A.1 Esquemes HBS

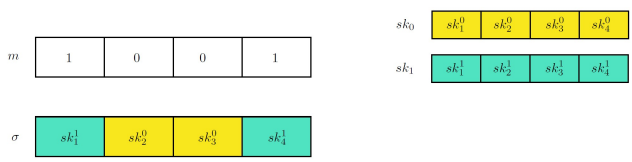

Fig. 8: Esquema de generació de claus Lamport. Font: *An Overview of Hash-Based Signatures* [5].

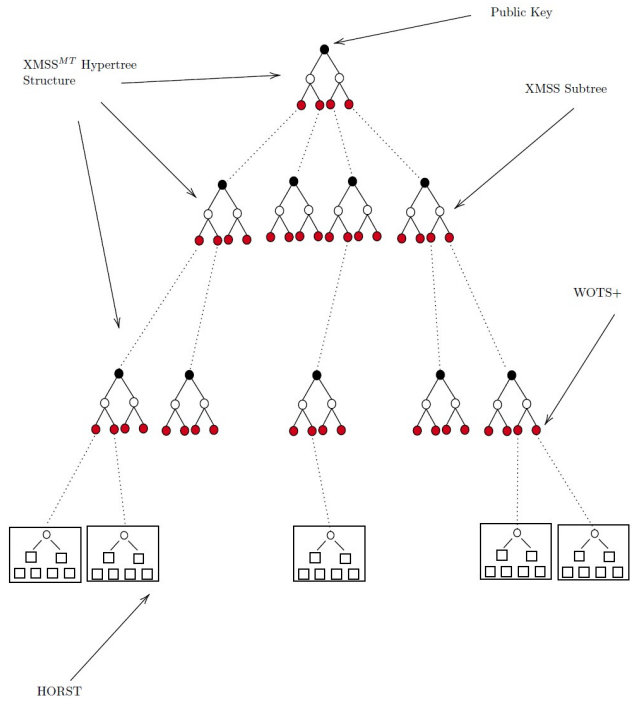

Fig. 9: Esquema de generació de claus Sphincs+. *An Overview of Hash-Based Signatures* [5].

A.2 Fragments de codi

```python
def generate_private_key_from_merkle_root(root):
    """
    Deriva una clau privada Bitcoin a partir del root utilitzant SHA-256.
    Args:
        root (bytes): L'arrel de l'arbre de Merkle que es fa servir com a entrada per generar la clau privada.
    Return:
        PrivateKey: La clau privada Bitcoin generada.
    """

    sk_hash = hashlib.sha256(root).digest()

    # Versió 0x80 per mainnet
    # Versió 0xEF per testnet
    version_byte = b'\xEF'

    payload = version_byte + sk_hash
    checksum = hashlib.sha256(hashlib.sha256(payload).digest()).digest()[:4]
    wif = base58.b58encode(payload + checksum).decode()

    return PrivateKey(wif)
```

Fig. 10: Codi de generació de les claus ECC en format Bitcoin. Font: Pròpia.

```python
def save_keys_to_files(private_key):
    """
    Guarda les claus pública i privada en arxius.
    Args:
        private_key (PrivateKey): La clau privada Bitcoin generada per guardar en un arxiu.
    """

    os.makedirs("ecc", exist_ok=True)

    with open("ecc/ecc_private_key_wif.txt", "w") as f:
        f.write(private_key.to_wif())

    public_key = private_key.get_public_key()
    with open("ecc/ecc_public_key_hex.txt", "w") as f:
        f.write(public_key.to_hex())
```

Fig. 11: Codi de guardat de les claus ECC en format WIF i hexadecimal. Font: Pròpia.

```python
def main():
    """
    Funció principal que coordina la generació de tots els components del sistema.
    No rep paràmetres. No retorna res.
    Les funcions que executa escriuen les seves sortides en fitxers .json o .txt
    segons el cas.
    """

    # 1. Generació de claus per a cada esquema HBS
    generate_lamport_keys()
    generate_wots_keys()
    generate_mss_keys()
    #generate_temp_sphincs_keys() #Nomes si s'utilitza windows
    generate_sphincs_keys()
    print()

    # 2. Construcció de l'arbre de Merkle i obtenció de l'arrel
    build_merkle_tree()
    print()

    # 3. Derivació de la clau privada ECC i guardat
    generate_ecc_keys_from_merkle_root()
    print()

    # 4. Generació de l'adreça Bitcoin i guardat
    generate_btc_address()

if __name__ == "__main__":
    main()
```

Fig. 12: Script d'automatització. Font: Pròpia.

```
pauhoms@PCpau:~/Desktop/TFG-PQC$ /bin/python3 /home/pauhoms/Desktop/TFG-PQC/pqc_generator.py
Claus Lamport generades i guardades en lamport/sk_Lamport.json i lamport/pk_Lamport.json
Claus WOTS generades i guardades en wots_plus/sk_Winternitz.json i wots_plus/pk_Winternitz.json
Claus MSS generades i guardades en mss_lots/sk_MSS.json i mss_lots/pk_MSS.json
Claus SPHINCS+ generades i guardades en sphincs/sk_Sphincs.json i sphincs/pk_Sphincs.json

Arbre Merkle creat i guardat.
Arrel: ec54413e10e853a572870f2e853c685435ba422904f9f08b4321f40e08700f7e

Clau privada i publica ECC Bitcoin generades i guardades correctament.

Clau Privada (WIF): cSDbekKrr9gETyqVwHyDHHCGT6eHjc55XGinNNzg8dhLRr26hsmp
Clau Publica (hex): 038e0c675b5b333e02648b5515b8976ff18484a2a78057d8d855b0479f593ee445
Direccio Bitcoin P2WPKH: tb1q0jzhlhczm65laa2d8z0lfjvhm5k70w5n035prx
```

Fig. 13: Output d'execució de pqc_generator.py. Font: Pròpia.

```python
def main():
    """
    Genera la clau pública corresponent,
    i crea una adreça Bitcoin SegWit (P2WPKH) per la xarxa testnet.
    """

    # Carregar la pk
    private_key = load_private_key()
    public_key = private_key.get_public_key()

    # Generar l'adreça P2WPKH
    address = public_key.get_segwit_address()

    with open("ecc/btc_address.txt", "w") as f:
        f.write(address.to_string())

    print("Clau Privada (WIF):", private_key.to_wif())
    print("Clau Publica (hex):", public_key.to_hex())
    print("Direccio Bitcoin P2WPKH:", address.to_string())
```

Fig. 14: Codi de generació de l'adreça Bitcoin en format P2WPKH. Font: Pròpia.

A.3 Altres

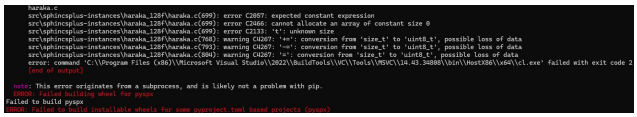

Fig. 15: Error a l'instalar el paquet pyspx en Windows. Font: Pròpia.