

HardVault: A Hybrid FPGA-Based Ethereum-Bitcoin Cold Wallet

Joel Poncha Lemayian¹, Member, IEEE, Ghyslain Gagnon², Senior Member, IEEE,
Kaiwen Zhang³, Member, IEEE, and Pascal Giard⁴, Senior Member, IEEE

Abstract—Cryptographic wallets play a vital role in securing digital assets within blockchain networks by managing private keys that authorize secure transactions. However, side channel analysis (SCA) attacks have become a serious threat, enabling attackers to extract sensitive information by exploiting algorithmic weaknesses in microcontroller-based wallets, resulting in the loss of millions of dollars in digital assets. In hierarchically deterministic (HD) systems, the compromise of a single primary key can endanger all subsequent child keys, while the use of independent keys for each account introduces complexity and challenges in key management. This work presents HardVault, a field programmable gate array (FPGA)-based cryptocurrency wallet that supports both Bitcoin and Ethereum. HardVault introduces the first hardware wallet architecture that implements both non-deterministic (ND) and HD key generation modes directly in hardware, giving users the flexibility to choose either approach based on their security and usability needs. By leveraging constant-time operations and hardware-enforced private-key isolation, the design significantly improves resilience to SCA attacks. In addition, the architecture prioritizes resource efficiency to minimize area usage without compromising security, making it well-suited for compact, portable hardware wallet applications. Implementation on a ZCU104 FPGA shows that HardVault uses only 27% of available look-up tables (LUTs). Compared to the Trezor One cryptocurrency (crypto) wallet, the proposed implementation achieves 9× higher energy efficiency, 8× lower latency, and 7× higher throughput.

Index Terms—Bitcoin, blockchain, cold wallets, cryptocurrency wallets, Ethereum, field programmable gate array (FPGA).

I. INTRODUCTION

BLOCKCHAIN networks heavily rely on cryptographic keys to ensure the security of user assets against malicious activities [1]. Major blockchain networks such as Ethereum and Bitcoin predominantly employ public-private key pairs to bolster network security [2]. They use cryptographic keys to generate digital signatures, authorize transfers, prove ownership, and create account addresses. While public

keys are openly accessible, the secrecy of private keys is paramount. They grant users full access to associated crypto assets [3], [4]. Consequently, a compromised private key can result in significant losses, with attackers potentially gaining control over all crypto assets within the compromised account. Blockchain users use crypto wallets to securely manage and monitor their cryptographic keys.

Crypto wallets are physical devices or software programs designed to generate and store public and private blockchain keys using cryptographic algorithms. crypto wallets are either hot (online and convenient but vulnerable) or cold (offline and more secure) [5]. They can also be hierarchically deterministic (HD), deriving many keys from a single primary key for easier management but risking a single point of failure, or nondeterministic (ND), generating independent keys that enhance security but complicate key management [5]. Regardless, wallet security remains a fundamental requirement across all types [3].

Several techniques have been proposed to strengthen cryptocurrency wallet security. For example, CryptoVault employs Intel SGX enclaves to manage keys securely [6], while the SBLWT wallet uses ARM TrustZone to protect private keys on mobile devices [7]. Similarly, commercial wallets such as Ellipal Titan [8], and COLDCARD [9] rely on “air-gapped” isolation to prevent unauthorized access.

The literature works discussed above propose HD crypto wallets implemented as software running on microcontroller units (MCUs). For instance, wallets that employ “air-gapped” isolation, such as the Ledger Nano X [10] and Trezor Model T [11], run on the STM32 microcontroller. However, these wallets are vulnerable to malware and side channel analysis (SCA) attacks [12], [13]. Studies show that these wallets can be compromised via malicious code injection during transaction signing [3].

On the contrary, implementing a crypto wallet directly in hardware can offer alternative security measures that provide stronger protection against malicious actors. Furthermore, integrating both HD and ND modes into the same wallet would allow users to select the complexity and security level that best matches their needs and risk tolerance, improving both usability and resilience against attacks.

A. Contributions

This work presents HardVault, the first complete hardware architecture and field-programmable gate array (FPGA) implementation of a hybrid Ethereum–Bitcoin cold wallet.

Received 8 December 2025; revised 18 April 2026 and 20 May 2026; accepted 20 May 2026. Date of publication 1 June 2026; date of current version 27 July 2026. This work was supported in part by the Mathematics of Information Technology and Complex Systems (MITACS) under Project IT28459 and in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) through Quantum eMotion Alliance under Grant ALLRP-588315. (Corresponding author: Joel Poncha Lemayian.)

Joel Poncha Lemayian, Ghyslain Gagnon, and Pascal Giard are with the Department of Electrical Engineering, École de technologie supérieure (ÉTS), Montréal, QC H3C 1K3, Canada (e-mail: joel-poncha.lemayian.1@ens.etsmtl.ca; pascal.giard@etsmtl.ca).

Kaiwen Zhang is with the Department of Software Engineering and IT, École de technologie supérieure (ÉTS), Montréal, QC H3C 1K3, Canada (e-mail: kaiwen.zhang@etsmtl.ca).

Digital Object Identifier 10.1109/TVLSI.2026.3696577

While prior work has implemented individual cryptographic primitives such as SECP256K1, hash-based message authentication code secure hash algorithm (HMACSHA)-512, Keccak, and Bitcoin improvement proposal (BIP)-32 derivation in hardware, no existing design integrates these into a unified wallet architecture that simultaneously supports both HD and ND key generation for both Ethereum and Bitcoin within a single hardware boundary. The main contributions of this work are as follows.

- 1) A unified hardware architecture supporting both Ethereum and Bitcoin, enabling dual-currency operation within a single FPGA-based wallet.
- 2) A cross-mode, cross-currency resource-sharing strategy that reuses cryptographic primitives across HD and ND modes and across Ethereum and Bitcoin simultaneously, reducing the total number of look-up table (LUT) utilized to 27% of the target FPGA—significantly below the sum of the individual mode implementations.
- 3) A configurable key-generation pipeline implementing both HD and ND schemes with a partial-path-caching mechanism that reduces per-key latency by approximately 40% for keys 2 through n in HD mode.
- 4) A comprehensive power-performance characterization demonstrating $9\times$ higher energy efficiency, $8\times$ lower latency, and $7\times$ higher throughput compared to the Trezor One commercial hardware wallet.

To guide the design, we set quantifiable targets, a logic utilization of approximately 65 k LUTs, and a minimum throughput of 10 kbps. This is based on analyses of existing FPGA cryptographic implementations and the performance requirements of current Ethereum and Bitcoin blockchains, as detailed in Section VI-E.

1) *Outline:* The remainder of this article is organized as follows. Section II provides the necessary background on HD and ND wallet types, as well as on the key algorithms they employ. Section III provides an overview of architectures from our prior work reused in this article. Section IV presents the proposed architecture of HardVault. Section V details the functional validation and verification of HardVault. Section VI reports FPGA implementation results and discusses real-world deployment, limitations, and future work. Finally, Section VII concludes this article.

II. PRELIMINARIES

This section provides background on the Ethereum–Bitcoin hybrid wallet. Notably, Sections II-A–II-C describe the HD wallet, ND wallet, and the address generation schemes, respectively. Sections II-D–II-J describe the algorithms ensuring security and integrity in key and address generation.

A. HD Wallet

As shown in Fig. 1(a), the HD crypto wallet comprises four primary parts: random number generation, mnemonic generation, seed generation, and child key derivation (CKD). A random number e is produced by a random number generator (RNG) and converted into mnemonic words mcs and a 512-bit seed sd using the BIP-39 standard [14]. The seed

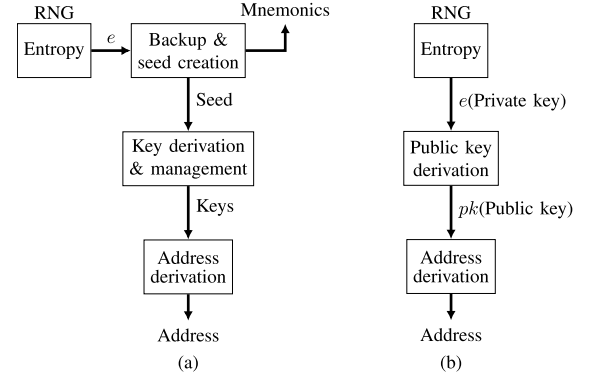


Fig. 1. Wallet generation workflows. (a) HD wallets use a primary seed for deterministic key derivation and unified backup, while (b) ND wallets generate each key-address pair independently of entropy.

sd is then processed by the CKD function, which employs HMACSHA-512 and SECP256K1 algorithms under the BIP-32 and BIP-44 standards to derive child private and public keys. Finally, the address-generation process uses hash functions such as Keccak and secure hash algorithm (SHA)-256 to generate addresses.

The BIP-32 standard, demonstrated in Fig. 2, defines a deterministic method for deriving child keys from a primary key using HMACSHA-512 and a CKD function, forming a hierarchical tree of blockchain addresses [15]. Its extension, BIP-44, specifies a six-level derivation path $m/purpose'/coin_type'/account'/change/address_index$, where m is the primary private key, and $address_index$ is a 32-bit index variable. The remaining levels are constants specific to each crypto. For each level, hardened (≥ 31 -bit, shown by apostrophe) and nonhardened (< 31 -bit) variables balance security, recovery, and convenience [14], [16]. The BIP-44 path for Ethereum is $m/44'/60'/0'/0'/address_index$, while that of Bitcoin is $m/44'/0'/0'/0'/address_index$.

B. ND Wallet

The ND crypto wallet generates only one pair of public and private keys. This means that all the key pairs are independent of each other. Consequently, a compromised key cannot be used to derive all other keys, as with a compromised primary key in a HD wallet.

Fig. 1(b) shows the process of generating keys in an ND wallet. Similar to the HD wallet, the first step for an ND wallet is to generate a random number e using an RNG. However, unlike the HD wallet, e is a private key. Second, SECP256K1 uses e to generate the corresponding public key pk .

C. Address Generation

Blockchain addresses are generated by hashing the public key through multiple functions. For Ethereum, the address A_{ETH} is computed as $KECCAK-256(K_{pub})$, where K_{pub} is the public key. The checksummed address $\hat{A}_{ETH}$ adds error detection by capitalizing certain letters based on the corresponding bits of a digest $D_A = KECCAK-256(A_{ETH})$. This mix of upper and lowercase letters prevents typing errors.

TABLE I
PARAMETERS FOR THE SECP256K1 EC

$p = 0x$ FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
$G = 0x$ 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8

Bitcoin, on the other hand, generates its address as $A_{BTC} = \text{RIPEMD-160}(\text{SHA-256}(K_{pub}))$ [17]. Its checksummed version $\hat{A}_{BTC}$ is formed by computing $D_A = \text{SHA-256}(\text{SHA-256}(A_{BTC}))$ and appending the first four bytes of D_A to A_{BTC} before encoding the result with the BASE58 encoding algorithm [17].

D. Password-Based Key Derivation Function-2

The password-based key derivation function-2 (PBKDF2) is a cryptographic key-derivation function used to generate a secure key from a password [18]. This algorithm uses a user-provided password and additional parameters to generate a distinctive key dk_{out} such that

$$dk_{out} = \text{PBKDF2}_{PRF}(Pwd, Sl, c, dk_{Len}) \quad (1)$$

where Pwd is the user-defined password, Sl is a salt variable used to further strengthen the security of the key, PRF is the preferred cryptographic hash function such as SHA-256 or SHA-512, c is the number of iterations, and dk_{Len} is the desired length of the output. HD crypto wallets use the PBKDF2 within the BIP-39 protocol to generate a mnemonic phrase and its corresponding seed value.

E. SECP256K1

SECP256K1 is a Weierstrass-form elliptic curve (EC) widely used in cryptos such as Bitcoin and Ethereum, defined by the following equation:

$$y^2 = x^3 + 7 \quad (2)$$

where x and y are coordinates on the curve. Other parameters that define SECP256K1 are shown in Table I, where p is a large prime number limiting operations to integers between 0 and $p - 1$, and G is the generator point given by $G = x04 \parallel x \parallel y$. SECP256K1 serves as the core algorithm used by Ethereum and Bitcoin wallets to generate public keys from private keys. Notably, using the elliptic curve point multiplication (ECPM) process that calculates a new point on the curve, as shown below

$$R = k \cdot P = \sum_{i=1}^k P \quad (3)$$

where k is a positive integer, and R and P are points on the EC.

F. Keccak Hash Function

Keccak is a secure hash function designed to resist collisions and SCA attacks [19]. Ethereum uses KECCAK-256 to hash public keys into addresses and to generate checksummed addresses, as described in Section II-B.

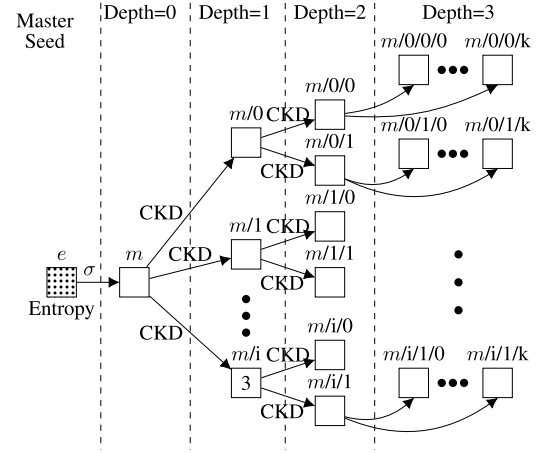


Fig. 2. BIP-32 HD key derivation structure showing primary seed, CKD functions, and multidepth child key generation with normal (m/i) and hardened (m/i') derivation paths, adapted from [15].

G. BASE58

BASE58 is a binary-to-text encoding algorithm designed for human readability by removing visually similar characters from its alphabet, these are “0,” “O,” “I,” and “l” [20]. Its 58-character set includes digits 1–9, 24 uppercase letters (excluding “I” and “O”), and 25 lowercase letters (except “l”). Bitcoin employs BASE58 to produce readable account addresses, reducing transcription errors and minimizing the risk of sending funds to incorrect destinations [21].

H. HMACSHA-512

Hash-based message authentication code (HMAC) is an algorithm proposed by the National Institute of Standards and Technology (NIST) to ensure data integrity and authenticity. HMACSHA-512 is defined as follows:

$$\text{HMAC}(k, m) = H(k \oplus opad \parallel H(k \oplus ipad \parallel m)) \quad (4)$$

where k and m are the secret key and message inputs of the algorithm, while $H(\cdot)$ denotes the SHA-512 cryptographic hash function. Ethereum and Bitcoin wallets use the HMACSHA-512 algorithm in various parts of their key generation process. In particular, it is used to generate the seed value, the primary key, and the child keys.

I. RIPEMD-160

RACE integrity primitives evaluation message digest (RIPEMD) is a family of cryptographic hash functions developed to enhance the security of message digest (MD) against collision attacks [22]. Among its variants, RIPEMD-160 is the most widely used. It processes an input message m of arbitrary length to produce a fixed 160-bit hash output [23]. As shown in Fig. 3, the algorithm features two parallel branches, left (A–E) and right (A'–E'), initialized with constants A_0 – E_0 . The message m is processed in 64-byte blocks divided into 16 four-byte words and iterated over 80 rounds, with functions F and variables I , k , and s updated every 16 rounds to ensure variability in the transformation process.

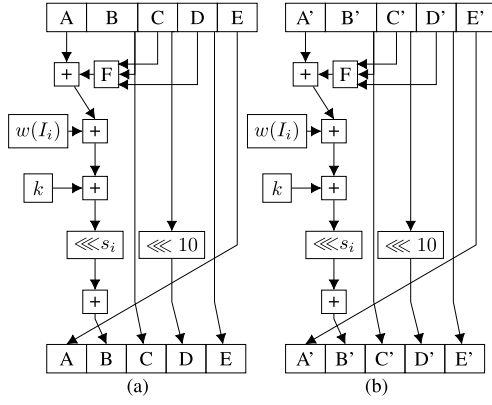


Fig. 3. One execution step of the RIPEMD-160 algorithm. A–E and A'–E' are registers. Moreover, the (a) left and (b) right sides are executed in parallel, and they process the same message block but with different constants for parameters F , i , k , and s . Adapted from [24].

After 80 rounds, the outputs of both branches are combined with the initial register values as follows [22]:

$$\begin{aligned}
 A_0 &= B_0 + C + D' \\
 B_0 &= C_0 + D + E' \\
 C_0 &= D_0 + E + A' \\
 D_0 &= E_0 + A + B' \\
 E_0 &= A_0 + B + C'.
 \end{aligned} \tag{5}$$

If m is split into multiple blocks, the new constants A_0 – E_0 become the initial values of the left and right branches when the next block is processed [24]. The final digest d_{ripemd} , is created by concatenating the results as follows:

$$d_{ripemd} = A_0 \parallel B_0 \parallel C_0 \parallel D_0 \parallel E_0. \tag{6}$$

The RIPEMD hash function is crucial for generating Bitcoin addresses within the Bitcoin wallet paradigm.

J. EC Digital Signature Algorithm

The elliptic curve digital signature algorithm (ECDSA) is a cryptographic algorithm based on EC arithmetic for generating digital signatures that ensure data integrity and authenticity. It is standardized by organizations such as the International Standards Organization (ISO), American National Standards Institute (ANSI), and NIST [25].

crypto wallets use ECDSA for transaction authorization, typically with the SECP256K1 EC. Hardware wallets handle key and signature generation internally to protect private keys during signing operations. The ECDSA signature generation equations are [26]

$$\begin{aligned}
 (x_1, y_1) &= k \cdot G \\
 r &= x_1 \bmod n \\
 s &= k^{-1} (z + d \cdot r) \bmod n
 \end{aligned} \tag{7}$$

where n is the order of the generator G , k is an ephemeral random integer in the interval $[1, n - 1]$, k^{-1} is the modular inverse of k , z is the SHA-256 digest of the message to be signed, d is the private key used for signing, and (r, s) is the signature.

K. CKD Function

As shown in Fig. 2, BIP-32 uses the CKD function, built upon SECP256K1 and HMACSHA-512, to derive child private and public keys in HD wallets. Algorithm 1 outlines this process, where inputs γ , δ , and λ represent the private key, chain code, and child number, respectively. Within the algorithm, the 256 most significant bits (MSBs) of the HMACSHA-512 digest form L , which yields the child private key $\hat{\gamma} = (L + \gamma) \bmod p$, while the 256 least significant bits (LSBs) form R , which becomes the new chaincode $\hat{\delta}$.

Algorithm 1 CKD Function as Described in BIP-32. Adapted From [15]

Input: γ, δ, λ

- 1: **if** $\lambda \geq 2^{31}$ **then**
- 2: $data \leftarrow 00 \parallel \gamma$ {Creation of hardened keys}
- 3: **else**
- 4: $p \leftarrow \text{SECP256K1}(\gamma)$
- 5: $data \leftarrow \text{SERIALIZE}(p)$
- 6: **end if**
- 7: $\hat{data} \leftarrow data \parallel \lambda$
- 8: $L, R \leftarrow \text{HMACSHA512}(\delta, \hat{data})$
- 9: $\hat{\gamma} \leftarrow \text{mod}(L + \gamma, p)$
- 10: $\hat{\delta} \leftarrow R$
- 11: **return** $\hat{\gamma}, \hat{\delta}$

III. ARCHITECTURE AND IMPLEMENTATIONS FROM PRIOR WORK

This work builds upon the SECP256K1, HMACSHA-512, CHECKSUM, and ECDSA hardware architecture introduced in our earlier studies [27], [28]. In particular, [27] presents a hardware architecture for SECP256K1 optimized for low resource utilization and enhanced security. It employs temporary registers to promote uniform read/write operations, thereby mitigating vulnerabilities to simple power analysis (SPA) and timing attacks. The assumed adversary is a passive observer who measures power consumption or electromagnetic emissions during ECPM to recover the private key bits k_i . Two structural countermeasures eliminate data-dependent variations by construction and facilitate constant-time behavior: uniform register access via R_i across both Montgomery Ladder branches, and branch-free complete addition formulas [29] that remove timing variability in elliptic curve point addition (ECPA) and elliptic curve point doubling (ECPD). The reader is referred to [27] for a detailed analysis. Furthermore, [28] introduces EthVault, an Ethereum HD cold-wallet architecture that integrates these cryptographic modules into a secure, resource-efficient framework.

The SECP256K1 implementation is specifically designed to enhance resistance against SPA and timing-based SCA attacks. This is achieved by using temporary registers to ensure uniform data handling during ECPM, and by applying the complete addition formula for ECPA. The use of complete formulas removes conditional branching in EC computations, variations that attackers often exploit to extract sensitive information [29].

The HMACSHA-512 module is implemented as a reusable component for multiple stages of a HD crypto wallet.

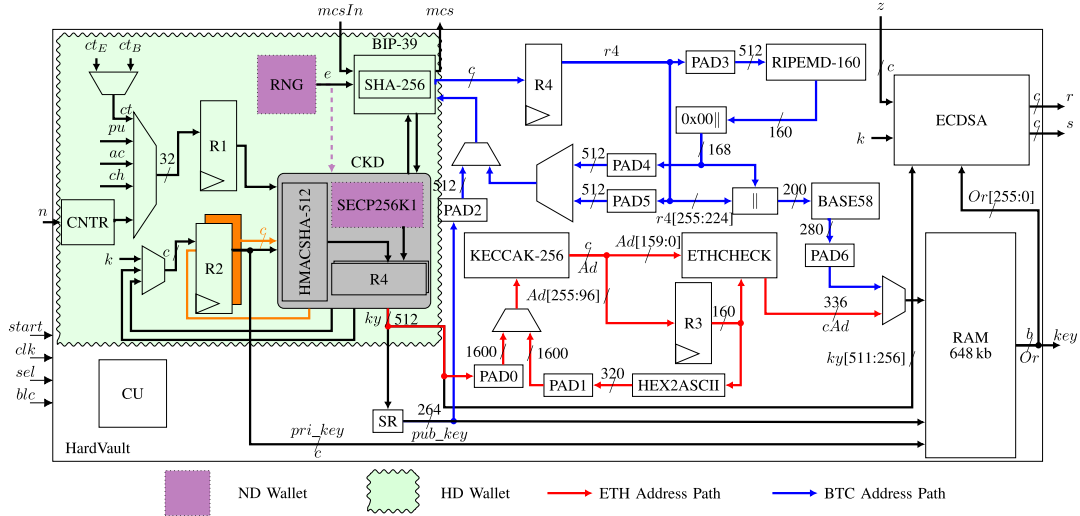


Fig. 4. Proposed hardware architecture of HardVault. The symbols pu , ct , ac , ch , and n are the parameters of a BIP-44 derivation path. ct_E and ct_B denote Ethereum and Bitcoin coin types, respectively. The variables $c = 256$, $b = 856$, and $key = Or[855:256]$.

To accommodate different stages, it includes additional inputs [k and m in (4)] sized according to the data requirements at each stage. Furthermore, the embedded SHA-512 core can be operated independently, enabling its use across various wallet processes.

The CHECKSUM module implements the Ethereum checksum algorithm (ETHCHECK in Fig. 4). It reuses a single KECCAK-256 core and applies offset-based conversion from hexadecimal to American Standard Code for Information Interchange (ASCII), eliminating the need for a LUT and thus reducing hardware resource usage.

Finally, the ECDSA module employs a modular multiplication algorithm to perform operations modulo n . To compute the modular inverse k^{-1} , it utilizes a binary inversion algorithm. Furthermore, the ECDSA utilizes the SECP256K1 within the child key derivation (CKD) module to compute $k \cdot G$.

While individual cryptographic primitives (SECP256K1, HMACSHA-512, ECDSA, and BIP-32 derivation) have been implemented in hardware in isolation, no prior work implements a unified hybrid ND+HD wallet supporting both Ethereum and Bitcoin in a single FPGA architecture. The novelty of HardVault lies in its cross-mode and cross-currency resource-sharing strategy, which reuses primitives across ND/HD modes and ETH/BTC paths simultaneously.

IV. PROPOSED HARDWARE ARCHITECTURE

This section presents the proposed hardware architecture of HardVault, shown in Fig. 4. In particular, Section IV-A discusses HardVault's overall functionality, including the generation of Ethereum and Bitcoin keys. Section IV-B discusses the wallet's execution flow. Sections IV-C–IV-E detail the proposed hardware architectures of the BIP-39, CKD function, and RIPEMD-160 algorithm, respectively. Furthermore, Section IV-F discusses the proposed hardware architecture of the BASE58 encoding algorithm.

A. Architecture of HardVault

Fig. 4 presents the overall architecture of HardVault. It supports both ND and HD key generation techniques. This

flexibility enables the user to prioritize either convenience or security. In the figure, the red and blue datapaths represent the address generation flows for Ethereum and Bitcoin, respectively. The dotted violet components and datapath illustrate the ND key generation process, while the black datapath corresponds to the HD public- and private-key generation process.

1) *Overview of HardVault Inputs and Outputs:* HardVault operates using a single synchronous clock signal (clk). The bld input determines the type of key to be generated.

- 1) 00 selects Bitcoin ND.
- 2) 01 selects Bitcoin HD.
- 3) 10 selects Ethereum ND.
- 4) 11 selects Ethereum HD.

The sel signal specifies which key to retrieve from random access memory (RAM) for signing, and $start$ initiates key generation. The parameter n is the number of keys generated per operation. The mcs output provides the mnemonic phrase, while $mcsIn$ enables the input of an existing mnemonic phrase for key recovery. The input z is the SHA-256 hash of the transaction to be signed, and outputs r and s are the ECDSA signature components. Finally, the key output delivers the public key and address selected by sel .

2) *ND Wallet Key and Address Derivation:* Fig. 1(b) shows that an ND wallet uses entropy from the RNG directly as the private key, from which the SECP256K1 algorithm derives the public key. This private-public key generation process is identical for both Ethereum and Bitcoin, though their address generation methods differ.

In Fig. 4, when the bld input is set to 00 for Bitcoin ND or 10 for Ethereum ND, the RNG generates a random number e , which serves as the private key and is stored in register R2. This private key is then processed by the SECP256K1 module within the CKD module to derive the corresponding public key. Subsequently, the wallet generates either a Bitcoin or an Ethereum address by processing the public key along the red or blue data path, respectively. Finally, the private key, public key, and address are stored in the RAM module for later use.

Resource efficiency is achieved by sharing a single SECP256K1 module across both Ethereum and Bitcoin public key generation, and reusing the SHA-256 module within the BIP-39 block for Bitcoin address derivation, eliminating two otherwise redundant hardware instantiations. Furthermore, the SECP256K1 instance used here inherits the SPA and timing-attack countermeasures described in Section III.

3) *HD Wallet Key and Address Derivation*: As described in Section II-A, the HD key generation process is more complex than the ND process because it utilizes various BIP protocols, including BIP-32, BIP-39, and BIP-44.

With HardVault, HD keys are generated by setting the *blc* input to 01 for Bitcoin HD or 11 for Ethereum HD. Before initiating the key generation process via the *start* signal, the user specifies the number of child keys to generate via *n*.

The process begins with the BIP-39 protocol, which converts the entropy *e* into mnemonic words (*mcs*) and a seed value using the SHA-256 and HMACSHA-512 algorithms. Rather than instantiating a dedicated HMACSHA-512 block, the proposed BIP-39 module reuses the existing instance in the CKD module via control signals, eliminating a redundant hardware instantiation and contributing to HardVault's compact footprint.

The BIP-32 architecture used for deriving child keys is identical for Ethereum and Bitcoin, although the BIP-44 constants used within BIP-32 differ. Fig. 2 depicts how the CKD function, utilizing the BIP-32 protocol, generates child keys. In Fig. 4, a single CKD module, comprising HMACSHA-512 and SECP256K1 algorithms, is employed. The constants in the figure represent the BIP-44 derivation path: *pu* (purpose), *ct* (coin type), *AC* (account), *ch* (change), and *n* (address index). A multiplexer selects between *ct_E* for Ethereum and *ct_B* for Bitcoin, with the coin type constants 60' for Ethereum and 0' for Bitcoin.

When generating multiple HD keys, the constants *m/purpose'/coin_type'/account'/change* of the complete path discussed in Section II-A are fixed, and only the address index varies across child keys. To improve throughput, the output of the CKD function for this partial path, when generating the first child key, is cached. This allows subsequent key generations to reuse the stored result instead of recalculating the partial path, significantly optimizing the private-to-public CKD process.

After the child keys are generated, the private, public, and address keys are stored in the RAM module.

4) *Transaction Authorization on HardVault*: To authorize a transaction, users must generate a digital signature using their private key. For security, the private key never leaves the wallet. HardVault integrates an ECDSA module to sign and authorize transactions on the blockchain.

Rather than transmitting the full transaction data (e.g., nonce, recipient address, and calldata) to the wallet, only the SHA-256 digest of this data is provided via the *z* input, reducing complexity and data size. When a transaction is to be signed, the *sel* input specifies which private key in RAM is used. The selected key is then routed to the ECDSA module via the *Or[255:0]* path.

During signing, the wallet internally generates the ephemeral value *k*, after which the ECDSA computation begins. Upon completion, the signature parameters *r* and *s* are output and used to validate the transaction on the blockchain.

B. Execution Flow and Key Generation Optimization

Fig. 5 illustrates the pipelined execution flow of HardVault, showing the sequential stages from entropy generation to transaction signing and the cryptographic algorithms applied at each stage. The blue blocks represent stages executed once per operation. In particular, entropy and mnemonic generation via BIP-39, and primary key and chaincode derivation via HMACSHA-512.

The orange blocks represent stages repeated *n* times in HD mode, where *n* is the number of child keys requested. Notably, child private and public key derivation using HMACSHA-512 and SECP256K1, address generation using KECCAK-256, RIPEMD-160, SHA-256, SHA-512, and BASE58, and finally transaction signing via ECDSA. The gray block represents the registers that cache the intermediate results of the partial BIP-44 derivation path, improving the latency of key generation by about 40%, as further discussed in Section VI-D.

C. Architecture of Bip-39

The BIP-39 module in Fig. 4 partially implements the BIP-39 protocol. This is because other algorithms utilized by the protocol are implemented within the CKD module. This protocol begins by using a random number generated from an RNG to produce a mnemonic phrase and a seed value. The mnemonic consists of 12 or 24 words that allow users to manage cryptographic keys in a HD wallet. The derived seed is then used to generate the primary key.

To perform this process, the BIP-39 protocol employs the SHA-256, SHA-512, and PBKDF2 algorithms. In particular, the PBKDF2 function relies on the HMACSHA-512 hash algorithm, as detailed in Section II-D.

Fig. 6 presents the hardware architecture of the BIP-39 module. The figure shows that the protocol uses the HMACSHA-512 hash algorithm within the CKD function and the SHA-512 within the HMACSHA-512 hash algorithm, using control signals.

In Fig. 6, *e* is the random number from the RNG, *mcs* is the generated mnemonic phrase, and *mcsIn* is the input for a mnemonic phrase to recover keys. Also, the red dotted region highlights the proposed hardware architecture of the PBKDF2 module. This module is a key component of the BIP-39 protocol, responsible for generating the seed value, as further described in Section II-A.

The MNG module in Fig. 6 stores the English wordlist defined by the BIP-39 standard [14]. It generates a sequence of mnemonic phrases, output through the signal *mcs*, using a random number *e* and an 8-bit checksum derived from the SHA-256 hash of *e*. Users are expected to record these mnemonics for backup and recovery purposes, and can later reenter them through the *mcsIn* input to regenerate the associated keys.

The mnemonic string should be padded with zeros to ensure the resulting bitstream is a multiple of 128 bytes, in alignment

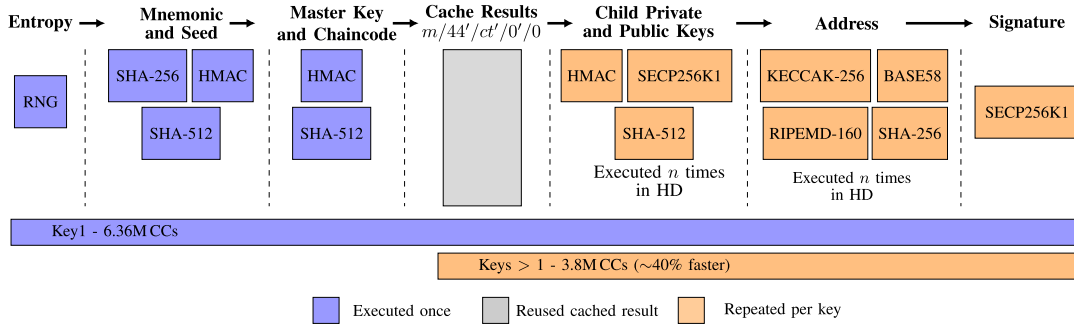


Fig. 5. Execution flow of HardVault, showing the sequential stages from entropy generation to transaction signing. The cryptographic algorithms used at each stage are indicated, where n denotes the number of child keys requested by the user in HD mode.

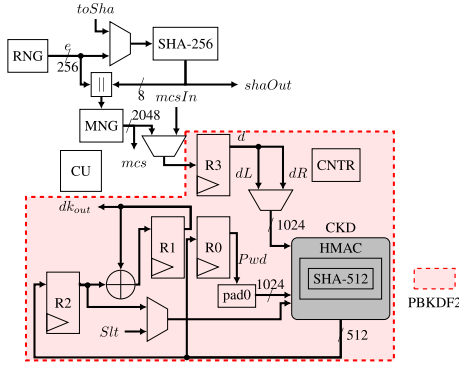


Fig. 6. Proposed hardware architecture of BIP-39 for seed and mnemonics generation. Uses the HMACSHA-512 module within the CKD function, and the SHA-512 within the HMACSHA-512 module, minimizing resource usage.

with the block size required by SHA-512 [14]. In the proposed architecture, the padded mnemonic data is extended to 2048 bits and stored in register R3, from which d is output. This 2048-bit value is then divided into two equal 1024-bit blocks ($dL = d[2047:1024]$ and $dR = d[1023:0]$), which are hashed using the SHA-512 function to produce a single digest. This digest is stored in register R0 and denoted as Pwd , as shown in Fig. 6.

In addition, the input salt Slt is formed by encoding the ASCII string “mnemonics” and padding it with zeros to 512 bits. This salt can optionally be concatenated with a user-defined personal identification number (PIN) to further enhance key security.

Once R0 is updated with the SHA-512 digest, the key-generation phase begins using the PBKDF2 algorithm. This involves performing 2048 iterations of HMACSHA-512, tracked by a counter module (CNTR). The intermediate digests are XORed together to form the final derived seed, which is stored in register R1 and denoted as dk_{out} , as defined by (1).

At the end of the BIP-39 process, this derived seed in R1 serves as the input for hierarchical CKD.

1) *Optimization*: The proposed architecture uses a single instance of the SHA-512 module within the HMACSHA-512 module, thereby avoiding duplication of hashing logic. Furthermore, the HMACSHA-512 module itself is reused within the CKD, with its operation managed by control signals.

In addition, the SHA-256 module is implemented as a standalone unit, configured to operate independently via the input

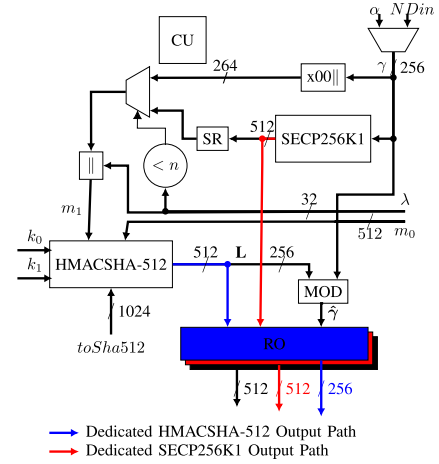


Fig. 7. Proposed hardware architecture of the CKD function. SR is the serialization of the public key to a compressed format. k_0 , k_1 , and $toSha512$ are 1024-bit inputs.

signal $toSha$ and the output signal $shaOut$. This configuration allows components or processes to leverage the SHA-256 functionality without duplicating hardware. For instance, the Bitcoin address path shown in Fig. 4 uses the SHA-256 module within the BIP-39 module.

By reusing cryptographic cores rather than instantiating multiple dedicated blocks, the architecture significantly reduces logic resource utilization, thereby improving efficiency and maintaining scalability.

D. Architecture of the CKD Function

Fig. 7 shows the proposed architecture of the CKD module that realizes the operations described in Algorithm 1. The algorithms utilized by the CKD function are also employed in other areas beyond the CKD function. For instance, the HMACSHA-512 algorithm is integral to the PBKDF2 in the BIP-39 protocol, where it is used to derive the seed. In addition, HMACSHA-512 is utilized to derive the primary private key and primary chain code outside the BIP-39 protocol and CKD function. Furthermore, the SHA-512 algorithm is employed both within and beyond the PBKDF2 in various capacities, as described in Section IV-C. Similarly, the SECP256K1 algorithm is used within the CKD function to compute the child public key and outside the CKD function to generate

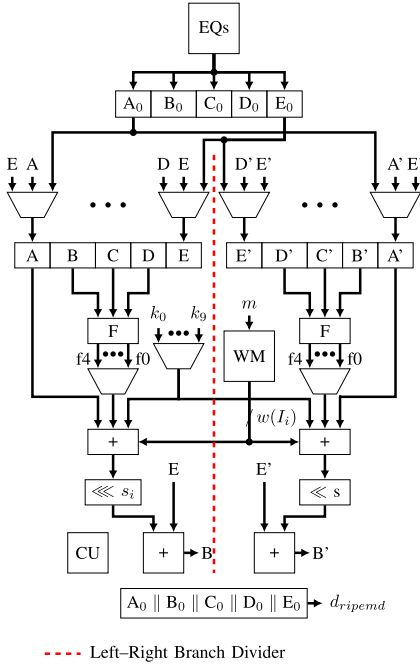


Fig. 8. Architecture of RIPEMD-160. To simplify the figure, some paths are not fully drawn. An arrow preceded or followed by a letter means a datapath from or to a register letter, e.g., $E \rightarrow$ and $\rightarrow B$.

regular keys. Also, the SECP256K1 algorithm is utilized by the ND wallet to create private keys.

1) *Optimizations*: To minimize the size of HardVault, we propose reusing the algorithms by modifying the CKD function and introducing the CKD architecture illustrated in Fig. 7. This design facilitates the reuse of each internal algorithm by providing dedicated input and output pathways. Inputs k_0 , k_1 , $toSha512$, and m_0 enable the wallet to utilize the HMACSHA-512 and SHA-512 algorithms outside the CKD module, with the digest stored in output registers R0. Moreover, the output of SECP256K1 also connects to the output register, allowing the standalone component to be utilized by external processes via input γ . A multiplexer connected to γ allows the RNG to send e directly to the SECP256K1 module through the $NDin$ input during ND wallet operation. In contrast, input α is reserved for other external processes.

E. Architecture of Ripemd-160

The proposed hardware architecture of the RIPEMD-160 module, which implements the RIPEMD-160 algorithm, is illustrated in Fig. 8. The design consists of two parallel computation branches, left and right. The branches use registers labeled A–E and A’–E’, respectively. Each register stores 4 bytes of data.

During the initialization phase of the hashing process, predefined constants stored in registers A_0 through E_0 are loaded into both branches. In HardVault, the input message to the RIPEMD-160 module is 64 bytes, as shown in Fig. 4. Consequently, the 64-byte message input m in Fig. 8 is divided into 16 words (w_0 – w_{15}) of four bytes each and stored in LUTs, denoted as WM. The RIPEMD-160 module, therefore, processes the words over a total of 80 rounds, as described in

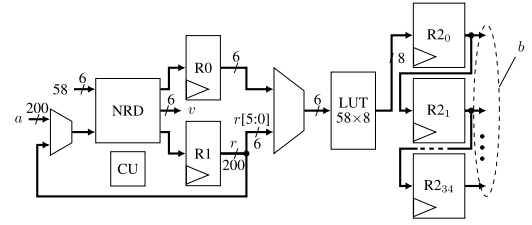


Fig. 9. Hardware architecture of the BASE58 encoder. The input a is iteratively divided by 58 using an NRD unit. The remainder selects a BASE58 symbol from the 58×8 LUT, which is shifted into R2 to form the encoded output b .

Section II-I. Within each 16-round sequence, the word $w(I_i)$ is selected based on the index I_i , which takes values from 0 to 15.

Each round uses the current register values, which are updated iteratively across rounds. The constants k_0 through k_9 correspond to the round-specific constants used in the 16-step transformations. In addition, the nonlinear function F operates on registers B, C, and D in the left branch, and on B’, C’, and D’ in the right branch, to generate round-dependent outputs.

After the 80 iterations (i.e., five rounds of 16 steps each), the EQs module computes updated values for A_0 – E_0 by applying the combination equations defined in (5). These updated values are then used to reinitialize the registers A–E and A’–E’, after which the same process is repeated using the second set of 16 input words (w_{16} – w_{31}), if w is greater than 64 bytes.

Finally, after completing the final set of 80 rounds, the RIPEMD-160 digest, denoted as d_{ripemd} , is produced by concatenating the final values stored in registers A_0 through E_0 , as specified in (6).

1) *Optimization*: As shown in Fig. 8, the left (A–E) and right (A’–E’) branches execute independent iterations. Both branches operate in parallel, enabling concurrent computation and reducing overall processing time.

F. Architecture of Base58

The BASE58 component in Fig. 4 corresponds to the BASE58 encoding algorithm described in Section II-G. The proposed hardware architecture is illustrated in Fig. 9. As discussed earlier, the BASE58 encoding process involves repeatedly dividing the input value by 58 and using each remainder as an index to select a symbol from the BASE58 alphabet. To efficiently perform this operation in hardware, the design employs a nonrestoring division (NRD) algorithm that performs division via iterative shift-and-subtract operations, thereby minimizing resource utilization.

The NRD block receives the input value a (a 200-bit Bitcoin address) and divides it by 58. The remainder and quotient are stored in registers R0 and R1, respectively. The remainder is then used as an index to access a symbol from the LUT that stores all 58 BASE58 symbols in ASCII format. The selected eight-bit symbol is then right-shifted into the R2 on each iteration.

R2 is a serial-to-parallel shift component formed by 35 cascaded eight-bit registers (R_{20} through R_{234}), yielding a 280-bit parallel output b . This width accommodates the worst

case encoding length: a 200-bit input produces at most 35 BASE58 symbols, each requiring eight bits of ASCII storage, for a total of $35 \times 8 = 280$. The control unit (CU) manages the encoding by repeatedly feeding the quotient from register R1 back into the NRD for successive division. This loop continues until the quotient is less than 58, after which both the final quotient and remainder are used as indices to obtain the corresponding symbols from the LUT.

Finally, the signal v from the NRD module indicates the number of valid bits or symbols accumulated in R2 at the end of the encoding process.

1) *Optimization*: To minimize resource utilization, the encoder employs a nonrestoring division algorithm that skips the restoration of the partial remainder after negative results. Instead, it compensates in the subsequent iteration by applying a correction, thereby reducing the number of arithmetic operations and simplifying the hardware implementation [32].

V. FUNCTIONAL VALIDATION AND VERIFICATION

This section discusses the functional validation and verification of the proposed architectures. Notably, it discusses the functional validation and verification of the BIP-39, CKD, RIPEMD-160, BASE58, and the HardVault wallet.

To validate the BIP-39 module, we first validated its algorithms. The PBKDF2 module was independently validated using a Python-based implementation, which generated diverse test vectors with varying passwords, salts, and iteration counts. The SHA-256 module was tested against official NIST vectors [33]. The complete BIP-39 implementation was verified using standard test vectors from [14] and additional entropy-mnemonic pairs from [34].

A Python-based reference model was developed to validate the functionality of the CKD hardware module. Starting from a predefined seed, the software generated a primary key and its corresponding chain code, and then derived multiple child private keys. These results were benchmarked against an online HD wallet generator for consistency and accuracy [34].

The CKD implementation was then evaluated by supplying the same primary key and chain code inputs. Its outputs, the derived child private and public keys, were compared against those of the software model. Core cryptographic modules (HMACSHA-512, SHA-512, and SECP256K1) were also tested in isolation using internal control signals. Robustness under edge conditions was validated, including both hardened and nonhardened derivation paths, as well as failure cases triggered by boundary issues in modular arithmetic operations and during HD/ND mode execution.

The RIPEMD-160 module was validated using a Python reference implementation. Test vectors, including edge cases of all zeros, all ones, and repeated patterns, were applied. The resulting hashes were compared against those produced by the hardware to ensure functional correctness, robustness, and full functional coverage.

To validate the functionality of the BASE58 architecture, a reference encoder was implemented in Python. This software model was used to generate test vectors that comprehensively covered various edge cases. In particular, tests included inputs such as zero, small integers (e.g., 1, 2, and 5), and large

values such as 2^{200} and all-one bit patterns. The outputs from the hardware encoder were then compared against the Python-generated results to verify correctness. In addition, the standardized test vectors provided in [20] were used further to confirm the accuracy and reliability of the proposed design.

For HardVault, random numbers representing e , generated by the RNG (as described in Fig. 1), along with the corresponding addresses from [35], are stored in a text file. A testbench reads this file during simulation, allowing the architecture to be thoroughly verified.

VI. FPGA IMPLEMENTATION RESULTS AND DISCUSSION

This section presents HardVault's FPGA implementation results. To the best of our knowledge, no comparable hardware-based crypto wallet architectures have been reported in the literature. Therefore, we compare HardVault with a commercial MCU-based wallet for context. Section VI-A lists the target platform and design tools, and Section VI-B describes the evaluation metrics. Section VI-C compares the proposed RIPEMD-160 module against state-of-the-art designs. Section VI-D presents full HardVault results, including BIP-39, CKD, and BASE58 modules. A performance comparison between HardVault and the commercial Trezor One hardware wallet is provided in Section VI-E, including power-consumption analyses. Finally, Section VI-F discusses real-world functionality and integration, while Section VI-G discusses limitations and future work.

A. Target Platform and Tools

A Xilinx ZCU104 Evaluation Kit, featuring a Zynq UltraSCALE+ (US) ZU7EV MPSoC, is used to implement the proposed HardVault architecture. The ZU7EV's programmable logic (PL) consists of 230 400 LUTs, 28 800 configurable logic blocks (CLBs), 460 800 registers, and 44.2 Mb of RAM. Moreover, it includes a processing system (PS) with a quad-core ARM Cortex-A53. We used Xilinx Vivado 2022.2 for design simulation, synthesis, and implementation. For power measurement, current and voltage probes were connected to the FPGA running HardVault, and a Tektronix TDS 3012 oscilloscope was used to visualize, analyze, and record the measurement data.

B. Metrics for Evaluation

Tables II–IV present implementation results on the ZCU104 FPGA. The “Platform” column in Table II lists the FPGA devices used in each reference. Although these platforms differ in capabilities and implementations, metrics such as LUT count provide a consistent estimate of area because their input sizes are similar across platforms. The area metric includes LUTs, digital signal processor (DSP) blocks, random access memory (RAM) blocks, and registers. Latency is measured in time and cc and throughput is calculated as follows [36]:

$$\text{Throughput (Mbps)} = \left(\frac{\text{Frequency (MHz)}}{\text{Latency (CC)}} \right) \times k \quad (8)$$

where k is the size of the output in bits. HardVault's power consumption is analyzed in Section VI-E, and compared against that of the Trezor One physical wallet [37], [38].

TABLE II
COMPARING IMPLEMENTATION RESULTS OF RIPEMD-160

Work	Platform	kLUT	DSP	Area RAM (kbits)	Registers	Frequency (MHz)	(ns)	Latency (clock cycle (CC))	Throughput ^a (Mbps)
This work	Zynq-US	1.48	0	0	358	250	320	80	500
Mehrabani [24]	Virtex-5	2.60	N/A	N/A	1 054	144	N/A	N/A	N/A
Knezevic et al. [30]	Virtex-II Pro	4.41	N/A	N/A	N/A	100	820	82	624
Sklavos et al. [31]	Virtex-II	16.11	N/A	N/A	1 600	73	N/A	N/A	2 000

^a Throughput calculated with (8), where $k = 160$.

TABLE III
RESOURCE UTILIZATION, PERFORMANCE, AND THROUGHPUT OF ETH/BTC HD-ND MODES VERSUS HARDVAULT

Modes	kLUT	Area RAM (kbits)	Registers	Frequency (MHz)	Latency (CC)	(ms)	Throughput ^a (kbps)
ETH HD	57.47	684	29 149	167	3 775 120	22.61	37.87
ETH ND	35.39	684	20 301	250	1 887 568	7.55	113.37
BTC HD	61.88	684	31 542	200	3 802 691	19.01	42.96
BTC ND	34.61	684	21 014	200	1 915 148	9.58	83.54
HardVault	63.71 (27%)	684 (6%)	32 922 (7%)	167	3 802 691	22.77	35.13

^a Throughput calculated with (8), where k is 856 and 800 for Ethereum and Bitcoin, respectively.

C. RIPEMD-160

Table II summarizes the implementation results of the proposed RIPEMD-160 architecture, depicted in Fig. 8, alongside comparable designs reported in the literature. Since some works omit specific performance metrics, comparative estimations are derived, where possible, using the available data.

The results indicate that the proposed implementation fits within the smallest hardware footprint, requiring only 1.48 kLUTs and 358 registers, with no DSP or RAM usage. It also attains the highest operating frequency of 250 MHz and achieves the lowest latency of 320 ns (80 CCs).

However, despite its advantages in area and latency, the proposed design yields the lowest throughput (500 Mbps) among the compared implementations, likely due to architectural tradeoffs favoring minimal resource utilization and low latency over sustained data rate. Nevertheless, the throughput remains sufficient for HardVault's requirements.

D. HardVault Hybrid Cold Wallet

Table III shows the implementation results of HardVault on the Zynq UltraSCALE+ (US) ZU7EV MPSoC FPGA. The table reports the resource requirements for the four individual wallet modes, which were developed independently. These are the Ethereum (ETH) HD, ETH ND, Bitcoin (BTC) HD, and BTC ND, as well as the unified HardVault architecture with the corresponding percentages of LUTs, RAM, and registers required.

The results indicate that HD wallets for both cryptos are more resource intensive than their ND counterparts. This increase stems from the additional cryptographic operations required for HD key derivation, such as the CKD function, which involves repeated HMACSHA-512 computations and multiple elliptic curve cryptography (ECC) operations. On the contrary, ND wallets bypass some of these algorithms and the iterative derivations, resulting in lower register usage, smaller LUT counts, and reduced latency.

Interestingly, the unified HardVault implementation does not simply accumulate the resources of all four variants. Instead, it demonstrates significant resource efficiency through extensive module reuse. For example, cryptographic building blocks such as the SHA-256, HMACSHA-512, and SECP256K1 modules are instantiated only once and shared across all modes and the stages shown in Fig. 5, with control logic determining their activation. This modular reuse strategy minimizes hardware redundancy, resulting in a total kLUT usage of 63.71, which is much lower than the sum of individual wallets.

The latency reported for HardVault, as shown in Table III, corresponds to the worst case scenario among the four modes, specifically the BTC HD wallet. The latency for the Ethereum HD and Bitcoin HD is the time required to generate the second child private-public key sets. However, as detailed in Section IV-A, subsequent key generations are significantly faster due to the storage and reuse of precomputed constants and the BIP-39 protocol, which is executed only once at initialization to generate the mnemonic phrase. In practice, the latency for generating the first key set is 6 356 729 CCs for Ethereum and 6 362 617 CCs for Bitcoin, whereas generating the second key sets requires only 3 775 064 CCs for Ethereum and 3 780 952 CCs for Bitcoin, corresponding to an average latency reduction of approximately 40%. These results demonstrate that HardVault achieves a balance between flexibility and hardware efficiency by supporting multiple wallet modes in a single architecture without incurring a proportional increase in resource utilization.

Table IV presents a breakdown of the resource utilization and latency for the primary cryptographic components implemented in HardVault. The table shows that the CKD module exhibits the highest demand for LUTs, registers, and latency. This high resource utilization is because the CKD integrates the SECP256K1 and the HMACSHA-512 components, as shown in Alg. 1.

The BIP-39 comprises the SHA-256 and the PBKDF2 components shown in Fig. 6. Its main area contributor is the

TABLE IV
RESOURCES UTILIZED BY HARDVAULT ON ZYNQ ULTRASCALE+ FPGA

Module	LUT	Registers	RAM (kbits)	Latency (CCs)	Latency (μ s) ^b
BIP-39	17 725	5 499	0	692 827	4 149
CKD	27 152	17 770	36	1 887 855 ^a	11 305
SECP256K1	21 059	13 873	0	1 887 520	11 303
ECDSA	10 148	4 606	0	1 888 550	11 309
HMACSHA-512	4 994	2 615	36	335	2.00
KECCAK-256	2 508	1 620	0	25	0.15
SHA-256	1 233	952	0	73	0.44
BASE58	1 166	908	0	27 340	164
RIPEMD-160	740	334	0	80	0.48
RAM	84	0	648	1	0.01

^a Time to create normal keys, 338 CCs for hardened keys.

^b Frequency of 167 MHz used.

MNG module (Fig. 6), which stores the English mnemonic word list [14] and accounts for 71.4% of its total LUT usage. Storing this list in external memory could greatly reduce on-chip utilization. Also, its high latency stems from the 2048 PBKDF2 iterations as discussed in Section IV-C, while CKD latency is dominated by the sequential execution of all the BIP-44 path elements required to produce the first child key.

Similarly, although the ECDSA algorithm relies on the SECP256K1 curve, the resources attributed to the ECDSA component do not include those consumed by the SECP256K1 module. This is because the SECP256K1 algorithm is executed within the CKD module rather than being directly embedded in ECDSA. As shown in Table IV, the ECDSA component requires approximately half the number of LUTs used by the standalone SECP256K1 module. Its latency, however, is slightly higher because, in addition to executing the SECP256K1 algorithm, the ECDSA performs modular arithmetic operations. As a result, the throughput of transaction signing in HardVault, as defined in (8), is 45.27 kbps, (Key size = 512).

In addition, other cryptographic components like HMACSHA-512 and KECCAK-256 contribute moderately to HardVault's resource usage. For example, HMACSHA-512 uses 8% of LUTs and registers required by HardVault, while KECCAK-256 uses 4% of LUTs and 3% of registers. These modules are invoked during key generation and message authentication.

Lighter hash functions, such as SHA-256, BASE58, and RIPEMD-160, show relatively minimal resource usage and latency. The RAM components require the least latency, as HardVault uses 1 CC to perform key read and write operations. Moreover, the RAM module uses the largest RAM because it stores all generated keys.

E. Comparison of HardVault Against the Trezor One Wallet

The Trezor One crypto physical wallet is among the earliest commercially available wallets that support multiple cryptos [37]. This section compares HardVault's throughput and power metrics against those of the Trezor One.

1) *Throughput Analysis*: Trezor One requires about 387 ms to execute the CKD function and a little over 94 ms to derive each element of the derivation path [39]. Thus, generating

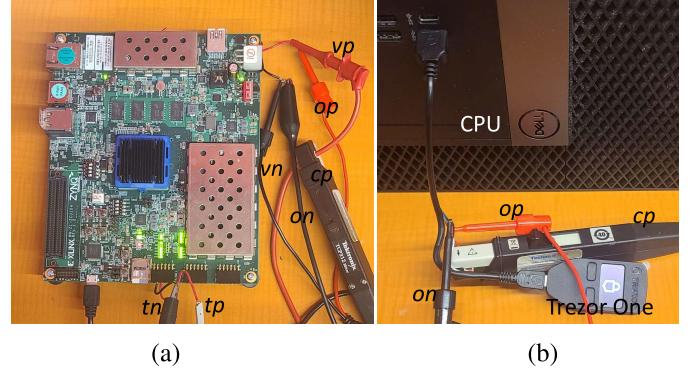


Fig. 10. Experimental setup. (a) FPGA board running HardVault with voltage and current probes. (b) Trezor one connected to the host CPU with probes capturing power during key generation.

a second private-public key pair takes little over 188 ms (two derivations: *address_index* and its corresponding public key) for both Ethereum and Bitcoin. In comparison, HardVault demonstrates over $8\times$ lower latency for the second key derivation. Moreover, the calculated throughput of Trezor One, $1/\text{latency}(\text{ms}) \times 856 = 5 \text{ kbps}$, suggests that HardVault achieves $\approx 7\times$ higher throughput.

In addition, the current Ethereum and Bitcoin blockchain networks exhibit rates ranging from 15–20 and 3–8 transactions per second (TPS), respectively, though these values fluctuate over time [40], [41]. Based on these rates, the required transaction signing throughput is at most 10 kbps for Ethereum and 4 kbps for Bitcoin (throughput calculated as $\text{TPS} \times (\text{size of ECDSA signature})$). Therefore, HardVault's 45.27 kbps transaction signing throughput easily supports both Ethereum and Bitcoin networks under current conditions.

2) *Power Analysis*: Following the throughput analysis, we evaluate the power efficiency of the proposed architecture to provide a comprehensive assessment of its performance. Fig. 10(a) shows the experimental setup used to measure the power utilization of the HardVault architecture implemented on a Xilinx ZCU104 FPGA. Fig. 10(b) depicts the power measurement setup for the Trezor One wallet. In both cases, a Tektronix TDS 3012 oscilloscope was used to calculate, visualize, and record the measurements.

For HardVault, the *clk* was configured to run at 167 MHz, generated using Vivado's Clocking Wizard, which was set to use the 300 MHz internal clock of the ZCU104 FPGA. The *start*, *rst*, and *btc* signals were connected to external switches, enabling convenient start, reset, and stop operations, as well as mode selection via *btc*. All other input/outputs (IOs) were connected to internal signals. Vivado's virtual input/output (VIO) and integrated logic analyzer (ILA) tools were used to drive and monitor these IOs in waveform form. A trigger port (*tp* and *tn*) was linked to the *start* signal and connected to the oscilloscope's trigger input, allowing power capture to begin precisely when HardVault starts generating keys. Power was supplied to the FPGA through *vp* and *vn*, while *cp* served as the current probe clamped around *vp*, and *op* and *on* as the voltage probes. The oscilloscope's math function was used to compute power by multiplying the current and voltage waveforms.

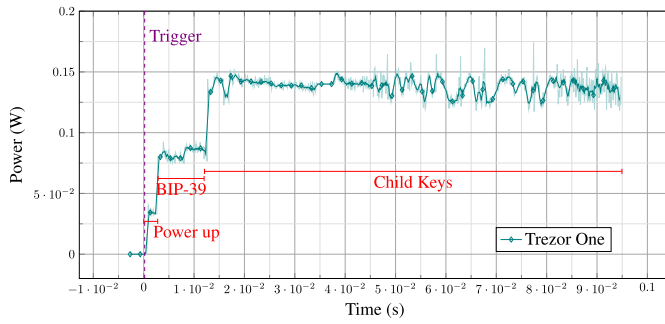


Fig. 11. Power consumption of trezor one during key generation: ≈ 90 mW for BIP-39 execution and ≈ 140 mW for CKD.

For the Trezor One setup (right side of Fig. 10), the universal serial bus (USB) cable was stripped to expose the positive and negative power supply lines. Voltage probes (*op* and *on*) were connected to measure the supply voltage, while a current probe (*cp*) was clamped around the positive supply line. Channel one of the oscilloscope, connected to the current probe, served as the trigger source.

The Trezor One wallet, containing both Ethereum and Bitcoin cryptos, was not protected by a passphrase and was connected to the central processing unit (CPU) with the Trezor Suite application running on the host computer to manage transactions. Once the CPU detected the Trezor One, it sent a request for a public key. Since no PIN protection was enabled, the wallet booted up, executed the BIP-39 protocol to derive the primary private key, and then generated the corresponding child keys [39].

Fig. 11 presents the measured power utilization of the Trezor One wallet. Multiple traces were captured using an oscilloscope, each exhibiting a similar overall profile. The plotted curve represents a centered moving average with a five-sample window, while the blurred background shows the raw data. Following the trigger event, three distinct power states are observed. The first, immediately after the trigger, corresponds to the wallet powering up. This is followed by a rise in power consumption, indicating the execution of the BIP-39 algorithm. A subsequent, more pronounced increase reflects the execution of computationally intensive algorithms, including SECP256K1, for deriving the child keys. The BIP-39 phase consumes approximately 88 mW and lasts about 10 ms, whereas the child key generation phase utilizes 140 mW.

Fig. 12 illustrates the measured power utilization of HardVault during the generation of Ethereum HD and ND keys, along with the reference power profiles of the Trezor One during its BIP-39 and child key computation phases. The HD power trace reveals two distinct stages. The first stage, characterized by a sharp rise in power immediately after the trigger, corresponds to the execution of the BIP-39 algorithm. This phase lasts approximately 4 ms, which aligns with the duration reported in Table IV.

The second stage corresponds to the CKD process. During this phase, power consumption increases further due to the execution of more computationally intensive operations, most notably the SECP256K1 algorithm used for child key generation. This stage extends to approximately 50 ms, after which

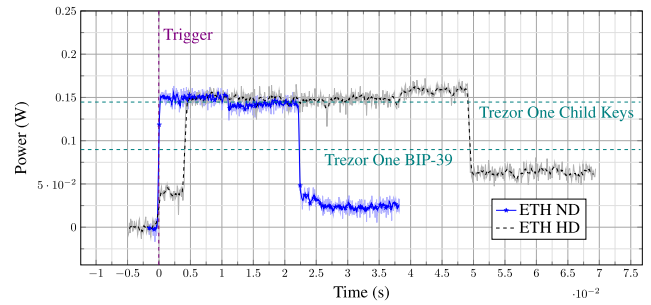


Fig. 12. HardVault Ethereum key generation power traces. HD: ≈ 40 mW (BIP-39) + ≈ 140 mW (derivation). ND: ≈ 140 mW (SECP256K1 only).

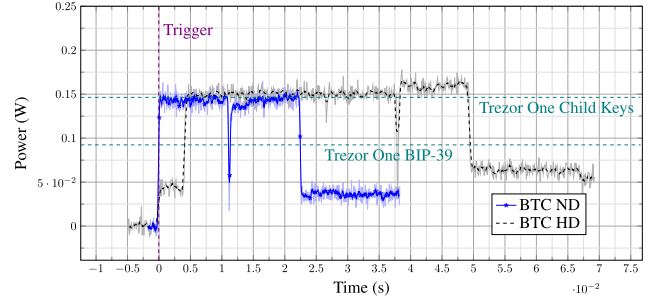


Fig. 13. HardVault bitcoin key generation power traces. HD: ≈ 40 mW (BIP-39) + ≈ 140 mW (derivation). ND: ≈ 140 mW (SECP256K1 only).

the power level drops sharply as HardVault enters an idle state. In this state, all cryptographic modules are reset, reducing switching activity and overall power consumption. The 50 ms period represents the total time required to generate the first key and complete a signing operation. Given that the number of CCs required to generate the initial Ethereum private–public key pair is 6356729, and the ECDSA operation requires 1888550 CCs (Table IV), the total duration can be confirmed as $(6356729 + 1888550) \text{ CCs} \times 1/167 \text{ MHz} = 49.37 \text{ ms}$, consistent with the observed power trace.

During the BIP-39 phase, HardVault utilizes approximately 40 mW, about 55% lower than the power used by the Trezor One. On the contrary, during the CKD phase, the average power consumption rises to roughly 140 mW, comparable to that of the Trezor One.

In the case of the Ethereum ND wallet, the power trace exhibits only a single active phase. Immediately after the trigger, the power consumption rises directly to approximately 140 mW, as this mode bypasses the BIP-39 algorithm. Instead, it proceeds directly to execute the SECP256K1 algorithm after initialization, generating the public key, computing the corresponding address, and signing. The subsequent drop in power reflects the transition to the idle state, during which all modules are reset to minimize switching activity. Notably, a distinct variation in power is observed around 11 ms, marking the point at which the CKD module completes the SECP256K1 execution, and the signing stage begins. This timing aligns with the results reported in Table IV.

Similar trends are observed in Fig. 13, which shows HardVault's power utilization during Bitcoin key generation. The HD trace contains two phases: the BIP-39 phase lasting ≈ 4 ms at 40 mW, followed by CKD at 140 mW. On the contrary, the ND trace has a single phase, where power rises

TABLE V
COMPARATIVE POWER EFFICIENCY ANALYSIS OF HARDVAULT
AND TREZOR ONE

Wallet	Power (mW)	Throughput (kbps)	EPO ($\mu\text{J/bit}$)
Trezor One	140	5	28
HardVault	140	45.27	3

directly to ≈ 140 mW after the trigger, as the wallet bypasses the BIP-39 stage and immediately executes SECP256K1 to compute the public key and address.

Reset points are more pronounced in the Bitcoin power trace than in the Ethereum curve. In the ND wallet, the SECP256K1 module is reset at around 11 ms before signing begins, while for HD, this reset occurs around 38 ms before signing. The Ethereum HD curve exhibits four distinct resets linked to CKD reinitialization as they prepare to process subsequent derivation data: approximately 16 ms after processing the *change* constant of the derivation path, 27 ms after executing the *address_index* constant, 38 ms following public key generation, and 49 ms after the signing phase.

The similarities in power utilization observed for HardVault during Ethereum and Bitcoin key generation are expected, as both blockchains employ comparable cryptographic techniques to derive keys, with the main difference being the address-generation methods. In addition, HardVault demonstrates significantly faster performance than Trezor One, completing the BIP-39 execution phase approximately $3\times$ faster, an advantage attributed to HardVault's hardware-accelerated design.

Furthermore, the energy per operation (EPO) metric, defined as Power/Throughput, is calculated for both the Trezor One and HardVault during child key generation. In particular, Trezor One exhibits an EPO value of $28 \mu\text{J/bit}$, whereas HardVault achieves a significantly lower value of $3 \mu\text{J/bit}$, as described by Table V. These results indicate that HardVault is $\approx 9\times$ more energy-efficient than the Trezor One wallet.

F. System Integration and Real-World Deployment

Section I explains that the derived addresses are accessible externally with a hot wallet to sign and authorize transactions while ensuring that sensitive credentials remain securely protected. This section outlines how we envision HardVault to interface with both a hot wallet and the end user.

Fig. 4 shows that the generated child private-public keys and their corresponding Ethereum and Bitcoin addresses are stored in a dedicated RAM module. To preserve security, only the public key and the derived address are accessible externally through the *key* interface. The private key never leaves the secure hardware boundary. These public values can be accessed via interfaces such as a USB port, a JTAG port, or an Ethernet port.

Communication with the hot wallet can be done through the USB interface, which transfers transaction data to the cold wallet for cryptographic signing. The FPGA then signs the transaction internally using secure digital logic modules, e.g., an ECDSA engine, and only returns the signed transaction. This design keeps the private key fully isolated and protected.

A display can also be employed during wallet initialization to present the 24-word mnemonic phrase via the *mcs* output.

For security reasons, including protection against physical tampering, these mnemonics are never stored on the device. Users are instructed to record them securely offline. During key recovery, the mnemonic phrase must be manually re-entered through the *mcsIn* input using an external interface. In practice, this interface can be implemented via a USB port. Notably, a standard USB keyboard or touchscreen can be connected to the ZCU104 PS via its USB host port, where an embedded driver captures the keystrokes and transfers them to the FPGA logic through an AXI interface.

G. Limitations and Future Work

HardVault is a proof of concept exploring the feasibility of the proposed approach; however, further work is needed before it can be made deployable. This section outlines current limitations and future work.

HardVault currently supports only Bitcoin and Ethereum, which utilize the same ECC curve (SECP256K1); future work will focus on extending support to additional cryptos and curves. Furthermore, HardVault is presently implemented on an FPGA; future work may explore an application specific integrated circuit (ASIC) implementation to improve energy efficiency for battery-operated wallets and potentially reduce cost through economies of scale as HardVault moves toward commercial deployment. Importantly, HardVault's modular, resource-sharing architecture, in which cryptographic primitives such as SECP256K1 and HMACSHA-512 are instantiated once and reused across all operating modes, is directly transferable to an ASIC implementation.

HardVault currently assumes an external secure entropy source and does not include an integrated entropy generator. Future work will address this limitation by incorporating a hardware-based solution, including a quantum random number generator (QRNG) developed by our collaborators [42]. In addition, further countermeasures against security threats beyond SPA and timing attacks will be investigated. A comprehensive and rigorous security analysis of the proposed countermeasures will also be conducted.

Keys generated by the wallet are currently stored in RAM blocks. In future versions, additional safeguards, such as encryption, application of a Zeroization path, retention mechanisms to preserve data in the event of power loss [storage in battery-backed RAM (BBRAM)], and error-correction techniques, will be considered to enhance security and reliability.

VII. CONCLUSION

This work proposed HardVault, a hybrid FPGA-based Ethereum-Bitcoin cold wallet that supports both ND and HD key-generation modes within a single hardware boundary. This dual-mode capability gives users the flexibility to choose between secure, one-time-use addresses (ND mode) and convenient multi-address derivation from a single seed (HD mode).

A cross-mode, cross-currency resource-sharing strategy reuses cryptographic primitives across all four wallet modes, achieving only 27%, 6%, and 7% of available LUTs, RAM blocks, and registers on the ZCU104 FPGA. A partial-path

caching mechanism further reduces per-key latency by $\approx 40\%$ for keys 2 through n in HD mode. Compared to the Trezor One hardware wallet, HardVault achieves $8\times$ lower latency, $7\times$ higher throughput, and $9\times$ better energy efficiency ($3\text{ }\mu\text{J/bit}$ vs. $28\text{ }\mu\text{J/bit}$).

REFERENCES

- [1] Y. Lu, "The blockchain: State-of-the-art and research challenges," *J. Ind. Inf. Integr.*, vol. 15, pp. 80–90, Sep. 2019.
- [2] S. Tikhomirov, "Ethereum: State of knowledge and research perspectives," in *Proc. Found. Pract. Secur.*, Feb. 2018, pp. 206–221.
- [3] M. Guri, "BeatCoin: Leaking private keys from air-gapped cryptocurrency wallets," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData)*, Jul. 2018, pp. 1308–1316.
- [4] J. P. Lemayian, G. Gagnon, K. Zhang, and P. Giard, "EVMx: An FPGA-based accelerator for smart contract processing," in *Proc. IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 34, Feb. 2026, pp. 554–567.
- [5] S. Suratkar, M. Shirole, and S. Bhirud, "Cryptocurrency wallet: A review," in *Proc. 4th Int. Conf. Comput., Commun. Signal Process. (ICCCSP)*, Sep. 2020, pp. 1–7.
- [6] N. Lehto, K. Halunen, O.-M. Latvala, A. Karinsalo, and J. Salonen, "CryptoVault—A secure hardware wallet for decentralized key management," in *Proc. IEEE Int. Conf. Omni-Layer Intell. Syst. (COINS)*, Aug. 2021, pp. 1–4.
- [7] W. Dai, J. Deng, Q. Wang, C. Cui, D. Zou, and H. Jin, "SBLWT: A secure blockchain lightweight wallet based on trustzone," *IEEE Access*, vol. 6, pp. 40638–40648, 2018.
- [8] (2025). *ELLIPAL Cold Wallet*. Accessed: Oct. 16, 2025. [Online]. Available: <https://www.ellipal.com/products/cold-wallet>
- [9] (2025). *Coldcard GitHub Repository*. Accessed: Aug. 12, 2025. [Online]. Available: <https://github.com/Coldcard>
- [10] *Ledger Nano S Plus: Secure and Manage Your Crypto and NFTs*. Accessed: Aug. 12, 2025. [Online]. Available: <https://shop.ledger.com/products/ledger-nano-s-plus>
- [11] (2025). *Trezor Model T: The Most Advanced Hardware Wallet*. Accessed: Aug. 12, 2025. [Online]. Available: <https://trezor.io/trezor-model-t>
- [12] M. S. Pedro, V. Servant, and C. Guillemet, *Side-Channel Assessment of Open Source Hardware Wallets*. Cryptology ePrint Archive. [Online]. Available: <https://eprint.iacr.org/2019/401>
- [13] K. Ngo and E. Dubrova, "Side-channel analysis of the random number generator in STM32 MCUs," in *Proc. Great Lakes Symp. VLSI*, Jun. 2022, pp. 15–20.
- [14] *BIP-0039: Mnemonic Code for Generating Deterministic Keys*. Accessed: Aug. 12, 2025. [Online]. Available: <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>
- [15] *BIP-0032: Hierarchical Deterministic Wallets*. Accessed: Aug. 12, 2025. [Online]. Available: <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>
- [16] (2025). *BIP-0044: Multi-Account Hierarchy for Deterministic Wallets*. Accessed: Aug. 12, 2025. [Online]. Available: <https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki>
- [17] A. M. Antonopoulos, *Mastering Bitcoin*. Sebastopol, CA, USA: O'Reilly Media, 2019.
- [18] H. Choi and S. C. Seo, "Optimization of PBKDF2 using HMAC-SHA2 and HMAC-LSH families in CPU environment," *IEEE Access*, vol. 9, pp. 40165–40177, 2021.
- [19] (2025). *Keccak: Team Keccak: Sponge, Permutation, Design*. Accessed: Oct. 16, 2025. [Online]. Available: <https://www.keccak.team/keccak.html>
- [20] M. Sporny, D. Longley, and D. Baird. (2021). *The Base58 Data Encoding*. [Online]. Available: <https://datatracker.ietf.org/doc/html/draft-msporny-base58-03>
- [21] M. Botta and D. Cavagnino, "A framework for reversible data embedding into Base45 and other non-Base64 encoded strings," *Appl. Sci.*, vol. 12, no. 1, p. 241, Dec. 2021.
- [22] H. Dobbertin, A. Bosselaers, and B. Preneel, "RIPEMD-160: A strengthened version of RIPEMD," in *Proc. Int. Workshop Fast Softw. Encryption*, 1996, pp. 71–82.
- [23] H. C. Van Tilborg and S. Jajodia, *Encyclopedia of Cryptography and Security*. Cham, Switzerland: Springer, 2014.
- [24] Y. Safaei Mehrabani, "Synthesis of an application specific instruction set processor (ASIP) for RIPEMD-160 hash algorithm," *Int. J. Electron. Lett.*, vol. 7, no. 2, pp. 154–165, Apr. 2019.
- [25] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *Int. J. Inf. Secur.*, vol. 1, no. 1, pp. 36–63, Aug. 2001.
- [26] B. Kieu-Do-Nguyen, C. Pham-Quoc, N.-T. Tran, C.-K. Pham, and T.-T. Hoang, "Low-cost area-efficient FPGA-based multi-functional ECDSA/EdDSA," *Cryptography*, vol. 6, no. 2, p. 25, May 2022.
- [27] J. Poncha Lemayian, G. Gagnon, K. Zhang, and P. Giard, "WiP: Towards a secure SECP256K1 for crypto wallets: Hardware architecture and implementation," 2024, *arXiv:2411.03910*.
- [28] J. P. Lemayian, G. Gagnon, K. Zhang, and P. Giard, "EthVault: A secure and resource-conscious FPGA-based Ethereum cold wallet," *IET Blockchain*, vol. 5, no. 1, p. 70028, Jan. 2025. [Online]. Available: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/blc2.70028>
- [29] J. Renes, C. Costello, and L. Batina, "Complete addition formulas for prime order elliptic curves," in *Proc. Ann. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2016, pp. 403–428.
- [30] M. Knezevic, K. Sakiyama, Y. K. Lee, and I. Verbauwhede, "On the high-throughput implementation of RIPEMD-160 hash algorithm," in *Proc. Int. Conf. Appl.-Specific Syst., Archit. Processors*, Jul. 2008, pp. 85–90.
- [31] N. Sklavos and O. Koufopavlou, "On the hardware implementation of RIPEMD processor: Networking high speed hashing, up to 2Gbps," *Comput. Electr. Eng.*, vol. 31, no. 6, pp. 361–379, Sep. 2005.
- [32] U. S. Patankar, M. E. Flores, and A. Koel, "Division algorithms—from past to present chance to improve area time and complexity for digital applications," in *Proc. IEEE Latin Amer. Electron Devices Conf. (LAEDC)*, Feb. 2020, pp. 1–4.
- [33] National Institute of Standards and Technology (NIST). (2025). *Cryptographic Algorithm Validation Program: Secure Hashing*. Accessed: Aug. 12, 2025. [Online]. Available: <https://tinyurl.com/mthefx3e>
- [34] *BIP39 Mnemonic Code Converter*. Accessed: Aug. 12, 2025. [Online]. Available: <https://iancoleman.io/bip39/>
- [35] M. M. Shabir and K. Zhang, "Qwallet: A hybrid cryptocurrency wallet using quantum RNG," in *Proc. 5th Int. Conf. Blockchain Comput. Appl. (BCCA)*, Oct. 2023, pp. 380–387.
- [36] M. A. Islam Romel, M. Rafiul Islam, and F. K. Fattah, "FPGA implementation of elliptic curve point multiplication for a 256-bit processor on NIST prime field," in *Proc. 14th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, Jul. 2023, pp. 1–7.
- [37] (2025). *Trezor Model One: The Original Crypto Hardware Wallet*. [Online]. Available: <https://trezor.io/trezor-model-one>
- [38] C. Guillemet, M. S. Pedro, and V. Servant, *Breaking Trezor One With Side Channel Attacks*. Accessed: Jul. 25, 2025. [Online]. Available: <https://www.ledger.com/th/blog/breaking-trezor-one-with-sca>
- [39] *Extracting the Private Key From a TREZOR*. Accessed: Aug. 12, 2025. [Online]. Available: <https://jochen-hoenicke.de/crypto/trezor-power-analysis/>
- [40] (2025). *The Future of Ethereum: What to Know in February 2025*. Accessed: Aug. 12, 2025. [Online]. Available: <https://www.forbes.com/sites/digital-assets/article/the-future-of-ethereum/>
- [41] *Bitcoin Performance Dashboard*. Accessed: Aug. 12, 2025. [Online]. Available: <https://chainspect.app/chain/bitcoin?range-1c=quarter>
- [42] B. Reulet, "Method for generating random numbers and associated random number generator," U.S. Patent 10 042,609, Aug. 7, 2018.