Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Audit-Report Coinbase cb-mpc Library Crypto 12.2024

Cure53, Dr.-Ing. M. Heiderich, Dr. N. Kobeissi, Dr. D. Bleichenbacher

Index

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

> *"Crypto creates economic freedom by ensuring that people can participate fairly in the economy, and Coinbase is on a mission to increase economic freedom for more than 1 billion people. We're updating the century-old financial system by providing a trusted platform that makes it easy for people and institutions to engage with crypto assets, including trading, staking, safekeeping, spending, and fast, free global transfers."*
>
> From https://www.coinbase.com/en-de/about

This report, assigned the unique reference ID CBS-02, presents the results and verdict of a cryptography audit and source code audit against the Coinbase cb-mpc library codebase, as performed by Cure53 in Q4 2024.

For background information, this security-centered initiative was requested by Coinbase Global, Inc. in April 2024 and follows an inaugural engagement targeting Coinbase cb-mpc library held in July of the same year (see report CBS-01). The evaluations for CBS-02 were performed by a three-person review team in November and early December 2024 (CW46-CW48). For maximum coverage and yield of findings, the client invested twenty-five days for analysis. All tasks for this procedure were placed into a single Work Package named *WP1: Cryptography audits & security assessments against Coinbase cb-mpc library code*.

Sources, test-supporting documentation, a detailed testing-priority roadmap, and any other assets deemed necessary to facilitate the undertakings were handed over to the Cure53 consultants in advance. This unfettered access was mandated by the selection of and conformance with a white-box pentest methodology. These items were also referred to during the preliminary phase, whereby Cure53 set some time aside during the week before the active evaluation window (i.e., CW45) to prepare the setup and foster a seamless start.

Communications between all relevant Coinbase and Cure53 personnel were enabled using a dedicated Mail thread. The discussions were efficient and productive, with minimal clarifying questions required. No delays or hindrances were encountered at any stage of the collaboration. Cure53 also performed live reporting of all pertinent discoveries using the aforementioned channel, encouraging swift remedial efforts.

The extensive examinations against the components in scope raised a total of nine negative circumstances for the internal team to consider. Only two were categorized as security vulnerabilities, while the remaining eight pertained to best practice alignments or general hardening measures.

The sum of tickets is generally moderate, which reflects positively on the security posture of the Coinbase cb-mpc library codebase. Even though only two exploitable vulnerabilities

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

were identified during this audit, one was graded with a *High* severity rating and should be resolved as soon as possible (see CBS-02-004).

Furthermore, Cure53 detected several general weaknesses and potential improvement vectors that would help to strengthen the Coinbase codebase if addressed.

As of January 6th, 2025, and in collaboration with the Coinbase cb-mpc team, Cure53 was able to verify that all outstanding security issues identified within this engagement were fully addressed.

All in all, Cure53 can state with confidence that the scrutinized project prioritizes crucial factors such as security and readability. Certain performance optimizations are viable, particularly concerning batch exponentiation, though these are not essential if the aforementioned assurances are consequently compromised. Application safeguarding and maintainability should remain the primary focus.

A number of key chapters are presented next, delineating various phases of the project as a whole. Firstly, the *Scope* documents all general insights in bullet point form, such as WPs, credentials, and any materials handed over by the internal team to facilitate the examinations. Next, the *Identified Vulnerabilities* and *Miscellaneous Issues* categories outline the security limitations observed by Cure53. These are provided in chronological order of detection and attach a high-level description, Proof-of-Concept (PoC) and/or steps to reproduce to verify the pitfall, and effective fix solutions. The report closes by summarizing Cure53's estimation of the researched features, discussing the construct's security posture and offering next steps for the client to action.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Cryptography audits & security assessments against Coinbase cb-mpc library code**
  - **WP1:** Cryptography audits & security assessments against Coinbase cb-mpc library code
    - **Source code:**
      - All relevant sources were shared in the form of .ZIP files with Cure53
        - Commit: *abcea5455c9d4f13735b0e3ce2c0361dee5f1420*
    - **High-priority coverage:**
      - EdDSA
      - Ec25519
      - Underlying libraries for the above
      - Cryptographic commitments
      - DRBG
      - Secret sharing
      - MPC protocols
      - Paillier
      - RSA
      - Elgamal
      - ECC
      - ZK
      - BigNum
      - Lagrange
    - **Low-priority coverage:**
      - RSA
      - PKI
      - Secp259k1
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *CBS-02-001*) to facilitate any future follow-up correspondence.

CBS-02-003 WP1: Missing check stipulated by *ECC-Refresh-MP* security proof *(Low)*

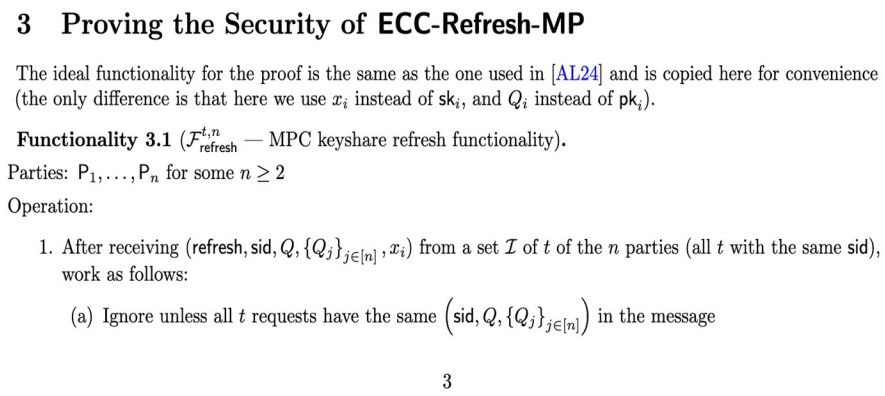
***Fix Note**: As of January 6th 2025, Coinbase has implemented a fix for this issue in the latest version of the Coinbase cb-mpc library, and this fix has been verified by Cure53.*

The *ECC-Refresh-MP* function is designed to refresh key shares in a multiparty setting for elliptic curve cryptography and is part of a suite of cryptographic protocols that support secure key management and transaction signing. For MPC protocols, all participating parties must agree on common parameters, such as session identifiers (*sid*), public keys (*Q*), and public key shares in order to ensure the correctness and security of the protocol execution.

The current implementation of the *ECC-Refresh-MP* function lacks a check that verifies whether all participating parties share the same session identifier, public key, and public key shares. Specifically, the function neglects to incorporate a mechanism that imposes a consensus on these parameters. This omission is acceptable in Coinbase's internal production code due to higher-level system protocols that ensure parameter consistency. However, in an open-source context whereby system-level assurances may not exist, the absence of these checks could lead to protocol deviations and potential security breaches.

The security proofs and formal analyses of MPC protocols typically assume that all honest parties agree on common parameters. The absence of parameter agreement checks in the implementation violates these assumptions, rendering theoretical security guarantees at least partially inapplicable. Specifically, the check stipulated in Section 3, 1.a of the *ec-dkg-theory.pdf* specification is absent from the code:

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de



3 Proving the Security of ECC-Refresh-MP

The ideal functionality for the proof is the same as the one used in [AL24] and is copied here for convenience (the only difference is that here we use x_i instead of sk_i, and Q_i instead of pk_i).

Functionality 3.1 ($\mathcal{F}^{t,n}_{\mathsf{refresh}}$ — MPC keyshare refresh functionality)**.**

Parties: $\mathsf{P}_1, \ldots, \mathsf{P}_n$ for some $n \geq 2$

Operation:

1. After receiving $(\mathsf{refresh}, \mathsf{sid}, Q, \{Q_j\}_{j\in[n]}, x_i)$ from a set $\mathcal{I}$ of t of the n parties (all t with the same sid), work as follows:
   (a) Ignore unless all t requests have the same $\left(\mathsf{sid}, Q, \{Q_j\}_{j\in[n]}\right)$ in the message

3

Fig.: Section 3, 1.a of ec-dkg-theory.pdf *specification.*

This situation raises numerous potential breach vectors. The first of those could allow an adversarial party to manipulate the shared parameters during protocol execution via parameter mismatch attacks. Without consensus checks, honest parties may unknowingly proceed with inconsistent views of the protocol state, leading to incorrect computations or leakage of sensitive information.

Secondly, replay and injection attacks are plausible here. An attacker could replay old messages or inject forged messages with altered parameters, causing the protocol to malfunction or disclose private key shares.

Affected file:
src/cbmpc/protocol/ec_dkg.cpp

Affected code:

```
// Spec-API: EC-Refresh-MP
error_t key_share_mp_t::refresh(job_mp_t& job, mem_t local_sid,
key_share_mp_t& current_key, key_share_mp_t& new_key) {
  error_t rv = 0;
  int n = job.get_n_parties();
  int i = job.get_party_idx();
  const crypto::pid_t& pid = job.get_pid();

  if (current_key.party_index != i) return coinbase::error(E_BADARG, "Wrong
role");
  if (current_key.Qis.size() != n) return coinbase::error(E_BADARG, "Wrong
number of peers");

  ecurve_t curve = current_key.curve;
  const mod_t& q = curve.order();
  const ecc_generator_point_t& G = curve.generator();
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
    }

    // Function continues without check…

}
```

To mitigate this vulnerability, Cure53 advises modifying the *ECC-Refresh-MP* function to include checks that ensure all participating parties hold the same *sid*, *Q*, and *Q_j* values. Alternatively, the Coinbase cb-mpc library documentation should clearly state that the function assumes all parties agree on shared parameters and that, without additional checks, the protocol may be insecure in certain environments. In addition, guidelines on implementing consensus mechanisms at the application layer should be provided if they are not incorporated into the Coinbase cb-mpc library functions.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

CBS-02-004 WP1: *ECC-Refresh-MP* vulnerable to small subgroup attacks *(High)*

***Fix Note**: As of January 6th 2025, Coinbase has implemented a fix for this issue in the latest version of the Coinbase cb-mpc library, and this fix has been verified by Cure53.*

In general, elliptic curve cryptography relies on the properties of an elliptic curve's prime-order subgroup in order to ensure security. All group elements involved in the protocol must belong to the correct subgroup to prevent attacks that exploit smaller subgroups within the curve. In an MPC context, validating the received elliptic curve points for subgroup membership guarantees that adversarial participants cannot introduce invalid or malicious points into the protocol execution. The continued failure to perform these checks can ultimately compromise the protocol's security guarantees.

The provided *ECC-Refresh-MP* implementation neglects to validate whether the received elliptic curve points, specifically the set *{R_{j, \ell}},* are valid group elements belonging to the prime-order subgroup. This validation step is explicitly required in the *ec-dkg-spec.pdf* reference specification under Step 3.a. on Page 8:

3. *Local instructions:* Each Party P_i, on receiving $\left(\{(c_{j,i}, r_{j,i}, \rho_{j,i})\}_{j\in[n]\setminus\{i\}}, \left\{\left(c_j, h_j, \{R_{j,\ell}\}_{\ell\in[n]\setminus\{j\}}, \rho_j\right)\right\}_{j\in[n]\setminus\{i\}}\right)$
   (a) Verify that all $\{R_{j,\ell}\}_{\ell\in[n]\setminus\{j\};\ j\in[n]\setminus\{i\}}$ are valid group elements (on the curve and in the subgroup)
   (b) For each $j \in [n] \setminus \{i\}$:
      i. Verify that $h_i = h_j$ and abort otherwise
      ii. Verify that $c_{j,i} = \text{Com-1P}(r_{j,i}, \mathsf{pid}_j, 1, \mathsf{pid}_i; \rho_{j,i})$ and abort otherwise
      iii. Verify that $c_j = \text{Com-1P}(\{R_{j,\ell}\}_{\ell\in[n]\setminus\{j\}}, \mathsf{pid}_j, 2; \rho_j)$ and abort otherwise
      iv. Verify that $R_{j,i} = r_{j,i} \cdot G$ and abort otherwise
   (c) *For all keys:*
      i. $x'_i \leftarrow x_i + \sum_{j<i}(r_{i,j} + r_{j,i}) - \sum_{j>i}(r_{i,j} + r_{j,i}) \bmod q$ for $j \in [n]$
      (We are assuming that the elements of the $[n]$ set have a fixed order.)
      ii. For $j \in [n]$:
         A. $Q'_j \leftarrow Q_j + \sum_{\ell<j}(R_{j,\ell} + R_{\ell,j}) - \sum_{\ell>j}(R_{j,\ell} + R_{\ell,j})$ for $\ell \in [n]$
      iii. Verify that $Q'_i = x'_i \cdot G$ and abort otherwise
      iv. Verify that $\sum_{j\in[n]} Q'_j = Q$, and abort if not
      v. Output $\left(x'_i, Q, \{Q'_j\}_{j\in[n]}\right)$

Fig.: Step 3.a. on Page 8 in reference specification.

Without this verification, an adversary could introduce points that do not belong to the subgroup, potentially exploiting small subgroup attacks to compromise the protocol. Viable attack vectors include a key compromise, whereby an adversary could force protocol participants to compute with points in a small subgroup, effectively leaking information about their private key shares. Furthermore, invalid points may cause the protocol to compute invalid results or fail altogether, leading to protocol disruption and a Denial of Service (DoS) for honest participants.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Lastly, certain security assumptions may be circumvented; small subgroup attacks break the discrete logarithm hardness assumption (ECDLP) upon which the security of ECC is based, rendering the protocol vulnerable to cryptanalysis.

Affected file:
src/cbmpc/protocol/ec_dkg.cpp

Affected code:

```
// Spec-API: EC-Refresh-MP
error_t key_share_mp_t::refresh(job_mp_t& job, mem_t local_sid,
key_share_mp_t& current_key, key_share_mp_t& new_key) {
  error_t rv = 0;
  int n = job.get_n_parties();
  int i = job.get_party_idx();
  const crypto::pid_t& pid = job.get_pid();

  if (current_key.party_index != i) return coinbase::error(E_BADARG, "Wrong
role");
  if (current_key.Qis.size() != n) return coinbase::error(E_BADARG, "Wrong
number of peers");

  ecurve_t curve = current_key.curve;
  const mod_t& q = curve.order();
  const ecc_generator_point_t& G = curve.generator();
  }

  // Function continues without check…

}
```

To mitigate this vulnerability, Cure53 suggests altering the corresponding code to ensure that all received elliptic curve points are validated for subgroup membership prior to protocol usage. This can be achieved via Coinbase cb-mpc library's existing *curve.check* functionality, which already implements all necessary checks:

```
error_t ecurve_t::check(const ecc_point_t& point) const {
  if (!point.valid()) return crypto::error("EC-point invalid");
  if (point.get_curve() != *this) return crypto::error("EC-point of wrong
curve");
  if (!point.is_in_subgroup()) return crypto::error("EC-point is not on
curve");

  if (!tls_allow_ecc_infinity) {
    if (point.is_infinity()) return crypto::error("EC-point is infinity");
  }
  return 0;
}
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

CBS-02-001 WP1: Ed25519 signing deviates from specification *(Info)*

***Fix Note**: As of January 6th 2025, Coinbase has implemented a fix for this issue in the latest version of the Coinbase cb-mpc library, and this fix has been verified by Cure53.*

Ed25519 is a widely adopted public-key signature system that offers high performance and strong security guarantees. One aspect of its security design is the deterministic generation of nonces, which are derived from a cryptographic hash of a private key prefix and the signed message. This approach aims to assert nonce uniqueness, preventing potential vulnerabilities associated with nonce reuse.

However, Coinbase cb-mpc library's implementation of Ed25519 deviates from the standard concerning nonce generation. Rather than derive the nonce deterministically from the private key and message, the code leverages OpenSSL's *RAND_bytes*. This approach effectively transforms the signature algorithm into a Schnorr signature scheme over the Ed25519 curve, with formatting adjustments to maintain compatibility with EdDSA verification processes.

Affected file:
src/cbmpc/crypto/ec25519_core.cpp

Affected code:

```
extern "C" int ED25519_sign_with_scalar(uint8_t* out_sig, const uint8_t*
message, size_t message_len, const uint8_t public_key[32], const uint8_t
scalar_bin[32]) {
  uint8_t nonce[64];
  RAND_bytes(nonce, 64);

  uint8_t az[32];
  for (int i = 0; i < 32; i++) az[i] = scalar_bin[31 - i];

  sign_with_nonce(out_sig, message, message_len, public_key, az, nonce);
  OPENSSL_cleanse(az, sizeof(az));
  return 1;
}

static void sign_with_nonce(uint8_t* signature, const uint8_t* message,
size_t message_len, const uint8_t public_key[32], const uint8_t az[32],
const uint8_t nonce[32]) {
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
  bn_t nonce_bn = from_le_mod_q(mem_t(nonce, 64));
  point_t R = point_t::mul_to_generator(nonce_bn);
  R.to_bin(signature);
  bn_t hram_bn = hash_hram(signature, mem_t(message, int(message_len)),
public_key);
  bn_t az_bn = from_le_mod_q(mem_t(az, 32));
  const mod_t& q = get_order();
  bn_t s = q.mul(hram_bn, az_bn);
  s = q.add(s, nonce_bn);
  s.to_bin(signature + 32, 32);
  mem_t(signature + 32, 32).reverse();
}
```

The signatures produced by this implementation are computationally indistinguishable from those generated by the standard EdDSA, except for the lack of determinism in nonce generation. Albeit, this deviation is not directly prone to risk as long as each message is signed only once and the nonce remains unique.

However, certain threats remain as a result of this alteration. Firstly, if the random nonce generation does not guarantee uniqueness (e.g., due to a flawed random number generator or insufficient entropy), nonce reuse could occur. Reusing a nonce in a Schnorr-like signature scheme can lead to private key recovery.

Secondly, the lack of deterministic nonce generation may prove problematic in systems that rely on deterministic signatures for auditability, reproducibility, and certain types of authentication protocols.

Finally, any deviation from the standard may introduce compatibility issues with other implementations expecting standard-compliant signatures, potentially facilitating verification failures.

To mitigate this issue, Cure53 suggests updating the Coinbase cb-mpc library documentation and clearly stating that the implementation employs a Schnorr signature scheme over the Ed25519 curve with formatting adjustments for EdDSA verification compatibility. The implementation's security assurances should be documented, emphasizing the importance of nonce uniqueness and the risks associated with nonce reuse.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

CBS-02-002 WP1: Variable time branching & recalculation in modular inversion *(Info)*

***Fix Note**: As of January 6th 2025, Coinbase has implemented a fix for this issue in the latest version of the Coinbase cb-mpc library, and this fix has been verified by Cure53.*

The *mod_t::_inv* function computes the modular inverse of an integer modulo to a prime *m*. Modular inversion is a fundamental operation in cryptographic algorithms such as those used in elliptic curve cryptography. Ensuring that these operations are optimized and performed in constant time is crucial for nullifying side-channel attacks and maintaining efficiency.

Two potential flaws persist in relation to *mod_t::inv*. Firstly, the function introduces variable-time behavior through the use of a conditional branch. The branching occurs based on the result of the *BN_mod_inverse* function, which checks whether the masked input (*masked_a*) and modulus (*m*) are co-prime. This introduces a timing variation that depends on the input data, potentially introducing timing side-channel attacks in function use locations.

Secondly, the masked value *masked_a* is computed and stored earlier in the code. However, the function recalculates its stored value (*mul(a, mask)*) later when calling *BN_mod_inverse*. While this evokes negligible security implications, performance optimizations can be administered here.

Affected file:
src/cbmpc/crypto/base_mod.cpp

Affected code:

```
void mod_t::_inv(bn_t& r, const bn_t& a) const {
  if (vartime_scope) {
    a.correct_top();
    auto res = BN_mod_inverse(r, a, m, bn_t::tls_bn_ctx());
    cb_assert(res && "vartime mod_t::inv failed");
  } else {
    bn_t mask = rand();
    bn_t masked_a = mul(a, mask);
    masked_a.correct_top();
    auto res = BN_mod_inverse(r, mul(a, mask), m, bn_t::tls_bn_ctx());
    if (!res) {  // The failure is likely due to masked_a and m are not co-
prime
      inv_mod_odd_const_time(r, a, m);
      return;
    }
    r = mul(r, mask);
  }
}
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

In certain scenarios such as handling private keys or nonces, timing side-channel attacks could compromise secret data confidentiality. Since modular inversions are ubiquitous in cryptography, this function may be utilized in a situation whereby timing attacks incur detrimental effects on security.

To mitigate this issue, Cure53 suggests replacing the variable-time branching with a constant-time approach, thus ensuring that the function in question cannot inadvertently interfere with security-critical functions elsewhere in the codebase.

CBS-02-005 WP1: Inconsistent function tagging between code/specification *(Info)*

***Fix Note**: As of January 6th 2025, Coinbase has implemented a fix for this issue in the latest version of the Coinbase cb-mpc library, and this fix has been verified by Cure53.*

Cure53 noted certain instances in the Coinbase cb-mpc library code whereby fundamental cryptographic functions that benefit from tag links to formal documentation descriptions (e.g., *// Spec-API: EC-Refresh-MP*) do not match the naming conventions used in the accompanying documentation or specifications (e.g., *ECC-Refresh-MP*). These inconsistencies can increase the difficulty of tracing the code back to the specification for developers, researchers, or reviewers, particularly when working in a multi-developer or open-source environment.

While this issue is evidently minor, resolving it will improve the clarity and maintainability of the Coinbase cb-mpc library. By standardizing naming conventions and aligning the code with the documentation, the development process will become increasingly seamless and intuitive for both internal and external parties.

CBS-02-006 WP1: Absent checks in Paillier key generation *(Medium)*

***Fix Note**: As of January 6th 2025, Coinbase has implemented a fix for this issue in the latest version of the Coinbase cb-mpc library, and this fix has been verified by Cure53.*

The Paillier cryptosystem relies on the mathematical properties of a modulus $N = p * q$, where p and q are large primes. To guarantee security and functionality, the following conditions must be met:

- ***p ≠ q:*** Ensures that the modulus N is not a perfect square and supports the system's underlying assumptions.
- ***gcd(N, (p - 1)(q - 1)) = 1*:** Ensures that the modular arithmetic used in Paillier operates correctly.

While the aforementioned checks are likely to hold with strong probability for randomly chosen primes, these checks are still considered mandatory for Paillier's security proofs to be valid and are standard across the vast majority of cryptographic implementations. Their

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

absence could undermine adherence to best practices and standards, particularly in environments where compliance or formal proofs are required.

Affected file:
src/cbmpc/crypto/base_paillier.cpp

Affected code:
```
void paillier_t::generate(int bits, bool safe) {
  int rv = 0;
  DYLOG_FUNC(LOG(bits), LOG(safe));

  p = bn_t::generate_prime(bits / 2, safe);
  q = bn_t::generate_prime(bits / 2, safe);

  N = p * q;

  update_private();
  has_private = true;
}
```

To mitigate this issue, Cure53 recommends integrating validation for the aforementioned checks, which will contribute to the construct's robustness, ensure adherence to security proofs, and guarantee compliance with industry standards.

CBS-02-008 WP1: Missing parameter checks in internal functions *(Info)*

***Fix Note**: As of January 6th 2025, Coinbase has implemented a fix for this issue in the latest version of the Coinbase cb-mpc library, and this fix has been verified by Cure53.*

Cure53 observed that certain internal functions lack parameter checks, which compromises the code's resilience to risk. While these omissions are non-exploitable at present unless calling functions elsewhere do not follow the necessary preconditions, the code will become more readable and easier to maintain if the checks are integrated.

Affected file #1:
zk/fischlin.h

Affected code #1:
```
struct fischlin_params_t {
  int rho, b, t;

  int e_max() const { return 1 << t; }
  uint32_t b_mask() const { return (1 << b) - 1; }
  void convert(coinbase::converter_t& c) { c.convert(rho, b); }  // t is
not sent
};
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Notably, *e_max* and *b_mask* can overflow if the *t* and *b* variables are larger than 32. Selecting parameters outside of the required range would result in proofs with reduced strength.

Affected file #2:
zk/small_primes.h

Affected code #2:

```
static error_t check_integer_with_small_primes(const bn_t& prime, int
alpha) {
  for (int i = 0;; i++) {
       int small_prime = small_primes[i];
       if (small_prime > alpha) break;
       if (mod_t::mod(prime, small_prime) == 0) return
coinbase::error(E_CRYPTO);
  }
  return 0;
}
```

To mitigate this issue, Cure53 advises ensuring that the caller of this function is aware of the maximal prime in *small_primes*. Since calling this function with a value alpha larger than the maximal prime leads to an out-of-bounds error, the value should be checked.

CBS-02-009 WP1: Batch exponentiation efficiency recommendations *(Info)*

While reviewing the code and corresponding documentation, Cure53 noted that some methods would benefit from optimization, particularly with regards to the computation of value:

$$P = \prod_i G_i c_i$$

Points G_i and scalars c_i can be computed using short addition sequences. Since constant time computation is unnecessary during the proof verification process, methods proposed by De Roij[1] and Pippenger[2] can be utilized.

A few locations whereby additional sequences can be leveraged for efficiency gains are outlined below; all proposed alterations would be local and would not require modifying the function headers.

Batch exponentiation can be used within the algorithm described in section 8.1.2 of *zk-proof.pdf* under the *Verification optimization* paragraph.

[1] https://iacr.org/cryptodb/data/paper.php?pubkey=2769
[2] https://cr.yp.to/papers/pippenger.pdf

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected file #1:
src/cbmpc/zk/zk_ec.cpp

Affected code #1:
```
error_t uc_dl_t::verify(const ecc_point_t& Q, mem_t session_id, uint64_t
aux)
      const {

  [...]

  bn_t z_sum = 0;
  bn_t e_sum = 0;
  ecc_point_t A_sum = curve.infinity();

  for (int i = 0; i < rho; i++) {
       bn_t sigma = bn_t::rand_bitlen(SEC_P_STAT);
       MODULO(q) {
             z_sum += sigma * z[i];
             e_sum += sigma * bn_t(e[i]);
       }
       A_sum += sigma * A[i];

       uint32_t h = hash32bit_for_zk_fischlin(
             common_hash, i, e[i], z[i]) & b_mask;
       if (h != 0)
             return rv = coinbase::error(E_CRYPTO,
             "invalid proof: zk_fischlin hash not equal zero");
  }

  if (A_sum != z_sum * G - e_sum * Q)
       return coinbase::error(E_CRYPTO, "invalid proof: A != z * G - e *
       Q");
  return 0;
}
```

Steps 5 and 6 of the *Verification optimization* described in *zk-proof.pdf*, section 8.4.1 can be modified with additional sequences.

Affected file #2:
src/cbmpc/zk/zk_elgamal_com.cpp

Affected code #2:
```
error_t uc_elgamal_com_t::verify(
      const ecc_point_t& Q, const elg_com_t& UV,
      mem_t session_id, uint64_t aux) const {

  [...]
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
  bn_t z1_sum = 0;
  bn_t z2_sum = 0;
  bn_t e_sum = 0;
  ecc_point_t A_sum = curve.infinity();
  ecc_point_t B_sum = curve.infinity();

  for (int i = 0; i < rho; i++) {

       [...]

       bn_t sigma = bn_t::rand_bitlen(SEC_P_STAT);
       MODULO(q) {
             z1_sum += sigma * z1[i];
             z2_sum += sigma * z2[i];
             e_sum += sigma * bn_t(e[i]);
       }
       A_sum += sigma * AB[i].L;
       B_sum += sigma * AB[i].R;

       [...]
}
```

Regarding section 8.7.2 in *zk-proof.pdf*, steps 5 and 6 of the verification algorithm described in the *Verification optimization* paragraph can be enhanced.

Affected file #3:
src/cbmpc/zk/zk_elgamal_com.cpp

Affected code #3:

```
error_t uc_elgamal_com_mult_private_scalar_t::verify(
       const ecc_point_t& Q, const elg_com_t& eA,
       const elg_com_t& eB, mem_t session_id, uint64_t aux) {

  [...]

  bn_t z1_sum = 0;
  bn_t z2_sum = 0;
  bn_t e_sum = 0;
  ecc_point_t A1_sum = curve.infinity();
  ecc_point_t A2_sum = curve.infinity();

  for (int i = 0; i < rho; i++) {
       if (rv = curve.check(A1_tag[i])) return rv;
       if (rv = curve.check(A2_tag[i])) return rv;

       bn_t sigma = bn_t::rand_bitlen(SEC_P_STAT);
       MODULO(q) {
             z1_sum += sigma * z1[i];
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
            z2_sum += sigma * z2[i];
            e_sum += sigma * bn_t(e[i]);
        }
        A1_sum += sigma * A1_tag[i];
        A2_sum += sigma * A2_tag[i];


        uint16_t h = hash32bit_for_zk_fischlin(
            common_hash, uint16_t(i), e[i], z1[i], z2[i]) & b_mask;
        if (h != 0) return coinbase::error(E_CRYPTO);
    }
    [...]
}
```

Furthermore, step (j) of the verification algorithm described in section 8.11.2 of *zk-proof.pdf* can be optimized with additional sequences.

Affected file #4:
src/cbmpc/zk/zk_pedersen.cpp

Affected code #4:

```
error_t range_pedersen_t::verify(
        const bn_t& q, const bn_t& g, const bn_t& h,
        const bn_t& c, mem_t session_id, uint64_t aux) const {

    [...]

    bn_t local_c_tilde[param::t];
    bn_t D = 0;
    bn_t F = 0;
    bn_t C = 1;

    bn_t c_tilde2[param::t];
    for (int i = 0; i < param::t; i++) {
         [...]

        MODULO(p) c_tilde2[i] = c_tilde[i] * c_tilde[i];

        bn_t rho_i = bn_t::rand_bitlen(64);

        MODULO(p_tag) {
            D += d[i] * rho_i;
            F += f[i] * rho_i;
        }

        bn_t c_tilde_c_ei = c_tilde2[i];
        MODULO(p) {
            if (ei) c_tilde_c_ei *= c;
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
                C *= c_tilde_c_ei.pow(rho_i);
            }
        }
        [...]
    }
```

The aforementioned code snippets pertain to areas whereby performance is critical and one can feasibly implement batch exponentiation without issuing alterations to the calling functions. Cure53 estimates that these functions can be accelerated by a factor approximately between 3 and 5, though the impact on overall performance remains unclear.

CBS-02-010 WP1: Typographical errors in documentation *(Info)*

***Fix Note**: As of January 6th 2025, Coinbase has implemented a fix for this issue in the latest version of the Coinbase cb-mpc library, and this fix has been verified by Cure53.*

Generally speaking, the implementation documentation is commendably composed, offering precise insights and facilitating straightforward comprehension of the corresponding source code. Necessary checks such as parameter verification are also included, which other conference papers on similar topics oftentimes omit.

Nonetheless, a small number of typos were detected in relation to cross references provided for reader context, as enumerated below:

- *zk-proofs-spec.pdf:* p.25, ZK-Two-Paillier-Equal-Interactive-2P step 4 (l): reduction should be (mod N_1) rather than (mod N_0)
- *ecdsa-2pc-theory.pdf:* The operations ⊙ and ⊕ are used without a definition.
- *ecdsa-2pc-spec.pdf:* Section 6.2: This identifier c'_{key} is used for both the encryption of x_1 and x'_1 under *N'.* The adoption of two distinct identifiers would be preferable.

To mitigate this issue, Cure53 suggests rectifying the minor errors in question via the guidance offered above.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

Coinbase cb-mpc is a library that implements highly sophisticated zero-knowledge and secure multi-party computation cryptography code in C++. The internal maintainers have paired clean and well-organized code with documentation of outstanding quality, enabling easier code understanding and seamless external evaluation. The protocol descriptions are precise and optimally structured to allow the creation of alternative implementations. The minor typographical errors described in ticket CBS-02-010 do not detract from the dev team's admirable efforts here, as no algorithmic errors or major omissions were discovered during Cure53's investigations.

The project's disciplined use of C++ and its exceptional documentation strongly facilitated the audit and deep dives into unusually complex cryptographic primitives. In summary for the notable findings, the library exhibited significant leeway for improvement, while a *High* severity vulnerability and numerous miscellaneous detriments were also encountered.

A two-phase exploration of the implemented protocols was conducted, the first entailing an assessment of the protocol documentation, which Cure53 verified for specification comprehensiveness. Contemporary conference papers typically omit insights into aspects such as parameter verification, for example, while the documentation in question is intended to closely reflect the implementation and hence includes checks and restrictions.

The second phase involved a holistic code evaluation and verification of the code's alignment with the specification.

Ticket CBS-02-003 confirms that the *ECC-Refresh-MP* function in the Coinbase cb-mpc library lacks a check to ensure that all participating parties agree on shared parameters, such as session identifiers, public keys, and public key shares, which is necessary for the correctness and security of MPC protocols. While this omission is acceptable in Coinbase's internal production due to higher-level assurances, it poses security risks in open-source environments, including parameter mismatch, replay, and injection attacks. Moreover, theoretical security assumptions could be violated.

In addition, *ECC-Refresh-MP* neglects to validate that received elliptic curve points belong to the correct prime-order subgroup, which is a critical step for preventing small subgroup attacks as outlined in the reference specification (see CBS-02-004). This omission could lead to significant disruptions in protocol security guarantees and a breakdown of the security assumptions underlying elliptic curve cryptography, explaining the upgraded impact score of *High*. Subgroup membership checks should always be integrated using Coinbase CoreCMS's *curve.check* functionality.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Elsewhere, Cure53 acknowledged that the Ed25519 signing in the Coinbase cb-mpc library deviates from the standard specification by using random rather than deterministic nonces, as highlighted in ticket CBS-02-001. This approach could introduce behavioral incompatibilities with standard Ed25519 implementations and expand the risk of nonce reuse.

Another weakness was observed whereby the modular inversion function in *mod_t::inv* introduces variable-time behavior due to conditional branching, potentially exposing the construct to timing side-channel attacks (see CBS-02-002). Additionally, this situation redundantly recalculates a stored masked value, which impacts performance.

Certain inconsistencies between function tags in the code and their corresponding names in the documentation were located and documented under ticket CBS-02-005, which increases the difficulty of tracing code back to specifications. Standardizing naming conventions will improve clarity and maintainability.

In addition, the Paillier key generation function in Coinbase cb-mpc library lacks essential algebraic checks that are mandatory for security proofs. These validations should be installed to strengthen security and assure adherence to best practices, as detailed in ticket CBS-02-006.

Cure53 also honed in on code clarity and reusability, since the project may be published as open source and reference implementation. These endeavors verified that the source code is ideally structured and the translation of protocols into code are clearly legible. Reusability means that individual functions can be repurposed without requiring the function caller to check preconditions. This condition is generally satisfied, though the careless use of internal functions could result in bugs, as enumerated in ticket CBS-02-008.

As of 6 January 2025, Cure53, working with the Coinbase cb-mpc team, has verified that all outstanding security issues identified during this engagement have been fully addressed.

Finally, Cure53 believes that several code functions can be accelerated by using batch exponentiation. The project generally prioritizes security over performance, considering that the code optimizations do not hinder the solution’s security effectiveness or readability. Nonetheless, ticket CBS-02-009 presents an enhancement proposal incurring minimal readability interference.

Cure53 would like to thank Eli Salm, Yehuda Lindell, Valery Osheter, Yi-Hsiu Chen, Arash Afshar, and Jeff Barksdale from the Coinbase Global, Inc. team for their excellent project coordination, support, and assistance, both before and during this assignment.