Alberta Buck - Formal Correctness Proofs (DRAFT v0.1)

Rerandomization and Re-Encryption

Perry Kundert

2026-04-17

The Alberta Buck identity system (Identity, Examples) rests on two cryptographic transformations: the *Identity Fountain* rerandomizes a Pointcheval-Sanders signature to produce an unlinkable credential, and the `approve` handshake re-encrypts the credential's ElGamal ciphertext under the counterparty's public key with a Chaum-Pedersen proof of equality. Everything downstream – bilateral transfers, the wormhole SNARK (Wormhole), Identity Guardian authorizations, regulatory recovery – inherits its correctness from these two primitives.

This document states and proves the correctness properties that the rest of the series relies on:

- **Theorem 1**: rerandomization preserves PS signature validity (completeness).
- **Theorem 2**: a uniform random rerandomization scalar yields a signature statistically identical to a fresh one, independent of the original (unlinkability).
- **Theorem 3**: an honest Chaum-Pedersen prover always verifies (completeness).
- **Theorem 4**: from two accepting transcripts with distinct challenges an efficient extractor recovers the witness (special soundness, 2-extractability).
- **Theorem 5**: the sigma-protocol transcript is perfectly simulatable given only public inputs and the challenge (HVZK).
- **Theorem 6**: combined, an accepted on-chain Chaum-Pedersen proof implies that the re-encrypted ciphertext carries the same issuer-certified identity point $M = m \cdot G$ as the registered credential (end-to-end re-encryption correctness).

Every theorem is accompanied by an executable `py_ecc` sanity check on BN254. The proofs assume only the q-SDH assumption (for PS signature unforgeability), the discrete logarithm assumption on alt_bn128 G_1, and the random oracle model (for the Fiat-Shamir transform) – all standard assumptions that Ethereum's security already requires. (PDF, Text)

Contents

1 Preliminaries

1.1 Notation

Symbol	Meaning
q	Prime order of G_1, G_2, G_T on BN254
G, g_2	Generators of G_1 and G_2
$e : G_1 \times G_2 \to G_T$	Bilinear, non-degenerate pairing (type-3)
O	Point at infinity (group identity)
$\mathbb{Z}_q^*$	Non-zero scalars mod q
$(x, y)/(X, Y)$	Issuer secret / public key: $X = xg_2,\ Y = yg_2$
$m \in \mathbb{Z}_q$	Identity scalar, $m = H(\text{identity\_data}) \bmod q$
$M = m \cdot G \in G_1$	Identity point (the value every ElGamal ciphertext encrypts)
$\sigma = (\sigma_1, \sigma_2) \in G_1^2$	PS signature: $\sigma_1 = h,\ \sigma_2 = (x + my)h$
(sk, pk)	Identity key pair on G_1: $pk = sk \cdot G$
$E = (R, C) \in G_1^2$	ElGamal ciphertext: $R = rG,\ C = M + r \cdot pk$
$H(\cdot)$	Cryptographic hash modelled as a random oracle (Fiat-Shamir)

All group operations in this document are written additively. BN254 has cofactor 1 on G_1, so every non-identity $P \in G_1$ has order q; there are no small-subgroup elements.

1.2 Hardness Assumptions

Three standard assumptions underpin every theorem in this document. None is specific to Alberta Buck – each one is already required by Ethereum's own security model or by any pairing-based construction built on the `ecPairing` precompile.

1. **Discrete Logarithm (DLog) on G_1**: given $P, Q \in G_1$, finding $\alpha \in \mathbb{Z}_q$ with $Q = \alpha P$ is infeasible. Protects ElGamal IND-CPA security, Schnorr unforgeability, and Chaum-Pedersen soundness.

2. **q-Strong Diffie-Hellman (q-SDH)**: the PS signature scheme is existentially unforgeable under chosen-message attack (EUF-CMA) assuming q-SDH[1]. We use this only as a **stated** assumption about the PS primitive; none of our proofs below reduce to it directly.

3. **Random Oracle Model (ROM)**: the Fiat-Shamir transform is applied to sigma protocols whose security is proved in the ROM, with the hash function H modelled as a random oracle bound to all public inputs and context (caller address, chain id, etc.).

The rerandomization proofs (Part I) are *unconditional*: Theorem 1 follows from bilinearity alone, and Theorem 2 is information-theoretic. The Chaum-Pedersen proofs (Part II) are interactive-sigma-protocol facts; the NIZK corollary (Part II, end) invokes the ROM.

[1] Pointcheval, D. and Sanders, O. "Short Randomizable Signatures." CT-RSA 2016, LNCS 9610, pp. 111-126. Theorem 4.2 establishes EUF-CMA security under the q-SDH assumption in bilinear groups.

2 Part I: Pointcheval-Sanders Signature Rerandomization

2.1 The Construction

A Pointcheval-Sanders signature on $m \in \mathbb{Z}_q$ is a pair $\sigma = (\sigma_1, \sigma_2) \in G_1^2$ with $\sigma_1 \neq O$ satisfying the pairing equation

$$e(\sigma_1,\ X + m \cdot Y)\ =\ e(\sigma_2,\ g_2).$$

The issuer produces σ by choosing a fresh $h \in G_1 \setminus \{O\}$ and setting $\sigma_1 = h,\ \sigma_2 = (x + my)h$. *Rerandomization* is the operation

$$\sigma'\ =\ (t \cdot \sigma_1,\ t \cdot \sigma_2) \qquad t \in \mathbb{Z}_q^*.$$

2.2 Theorem 1 (Completeness of Rerandomization)

Statement. For every valid PS signature σ on m under issuer key (X, Y), and for every $t \in \mathbb{Z}_q^*$, the pair $\sigma' = (t\sigma_1, t\sigma_2)$ is also a valid PS signature on m under the same issuer key.

Proof. By bilinearity of e:

$$\begin{aligned}
e(\sigma_1',\ X + mY) &= e(t\sigma_1,\ X + mY) && \text{(definition)}\\
&= e(\sigma_1,\ X + mY)^t && \text{(bilinearity)}\\
&= e(\sigma_2,\ g_2)^t && \text{(original validity)}\\
&= e(t\sigma_2,\ g_2) && \text{(bilinearity)}\\
&= e(\sigma_2',\ g_2).
\end{aligned}$$

Non-degeneracy: G_1 has prime order q, so $t\sigma_1 = O$ iff $t \equiv 0 \pmod q$ or $\sigma_1 = O$. Both are excluded by hypothesis ($t \in \mathbb{Z}_q^*$ and $\sigma_1 \neq O$ by PS validity). Therefore $\sigma_1' \neq O$ and the pair satisfies the full PS validity predicate. ■

Implementation guard. The registration contract MUST enforce $\sigma_1' \neq O$. Without this check, a prover can submit $\sigma' = (O, O)$ (obtainable by picking $t = 0$); the pairing equation then reduces to $1 = 1$ in G_T *for every* m, allowing a fabricated credential to pass registration. Theorem 1's completeness argument explicitly depends on $t \neq 0$; the contract's guard is the runtime witness of this hypothesis.

2.3 Theorem 2 (Statistical Unlinkability)

Statement. Let σ be any fixed valid PS signature on m. If t is sampled uniformly from $\mathbb{Z}_q^*$, then the distribution of $\sigma' = (t\sigma_1, t\sigma_2)$ is identical to the distribution of PS.Sign(sk_{issuer}, m) for h sampled uniformly from $G_1 \setminus \{O\}$, and is *independent* of σ.

Proof. Since G_1 is cyclic of prime order q and $\sigma_1 \neq O$, the map

$$\varphi_\sigma : \mathbb{Z}_q^* \to G_1 \setminus \{O\}, \qquad t \mapsto t\sigma_1$$

is a bijection (it is the restriction to non-zero scalars of the group isomorphism $\mathbb{Z}_q \to \langle \sigma_1 \rangle = G_1$). Hence σ_1' is uniform on $G_1 \setminus \{O\}$ regardless of the specific σ_1 we started with.

Given $\sigma_1' = h'$, the second component is forced: $\sigma_2' = t\sigma_2 = t(x + my)\sigma_1 = (x + my)(t\sigma_1) = (x + my)h'$. That is exactly what a fresh PS.Sign with random h' would output.

So the joint distribution of σ' matches the fresh-signature distribution, and the map $\sigma \mapsto \sigma'$ destroys all information about σ beyond the fact that both sign the same m. This is *statistical* (information-theoretic) unlinkability, not merely computational. ■

Remark. The argument uses only that G_1 has prime order; no cryptographic assumption is invoked. A cofactor > 1 would leak subgroup membership of σ_1; the prime-order property of BN254 G_1 is therefore load-bearing.

2.4 Sanity Check (py_ecc on BN254)

```
import secrets, hashlib
from py_ecc.bn128 import bn128_curve as bc
from py_ecc.bn128 import pairing

ORDER = bc.curve_order
G1, G2 = bc.G1, bc.G2
multiply, add, neg, eq = bc.multiply, bc.add, bc.neg, bc.eq

def rand():
    return secrets.randbelow(ORDER - 1) + 1

# Issuer key pair
x, y = rand(), rand()
X, Y = multiply(G2, x), multiply(G2, y)

# Alice's identity and PS signature sigma = (h, (x + m y) h)
m = int(hashlib.sha256(b"Alice Johnson | Alberta | AIC-2026").hexdigest(), 16) % ORDER
h = multiply(G1, rand())
sigma = (h, multiply(h, (x + m * y) % ORDER))

# Rerandomize with t != 0
t = rand()
sigma_p = (multiply(sigma[0], t), multiply(sigma[1], t))

# --- Theorem 1: completeness ---
lhs = pairing(add(X, multiply(Y, m)), sigma_p[0])
rhs = pairing(G2, sigma_p[1])
assert lhs == rhs, "Theorem 1 FAILED: rerandomized signature does not verify"
assert not eq(sigma_p[0], bc.Z1), "Theorem 1 guard: sigma'_1 == 0"
print("Theorem 1 (completeness):           PASS")

# --- Theorem 2: fresh rerandomization looks like a fresh signing ---
# Two independent rerandomizations should be indistinguishable from two
# independent fresh signings.  We check both give valid signatures on m
# whose sigma_1 components are independent uniform-looking elements.
t2 = rand()
sigma_p2 = (multiply(sigma[0], t2), multiply(sigma[1], t2))
assert not eq(sigma_p[0], sigma_p2[0]), "distinct rerandomizations collide"
lhs2 = pairing(add(X, multiply(Y, m)), sigma_p2[0])
rhs2 = pairing(G2, sigma_p2[1])
assert lhs2 == rhs2
print("Theorem 2 (unlinkability):          PASS  (two rerandomizations both verify, distinct)")

# --- Theorem 1 guard: the trivial signature (t = 0) is degenerate ---
zero_sig = (bc.Z1, bc.Z1)
lhs0 = pairing(add(X, multiply(Y, m)), zero_sig[0])   # pairing with 0 -> 1_GT
rhs0 = pairing(G2, zero_sig[1])                       # pairing with 0 -> 1_GT
# The pairing equation holds for ANY m -- this is why the guard matters
assert lhs0 == rhs0
# But the registration contract rejects it because sigma'_1 == 0:
rejected = eq(zero_sig[0], bc.Z1)
assert rejected
print("Theorem 1 guard (sigma'_1 != 0):    PASS  (trivial signature rejected by guard)")


Theorem 1 (completeness):           PASS
```

```
Theorem 2 (unlinkability):           PASS  (two rerandomizations both verify, distinct)
Theorem 1 guard (sigma'_1 != O):     PASS  (trivial signature rejected by guard)
```

How to read the output: Three PASS lines. The first confirms Theorem 1's pairing identity on a random rerandomization. The second confirms two fresh rerandomizations are distinct and both verify (a minimal demonstration of the unlinkable-yet-valid property). The third demonstrates *why* the $\sigma'_1 \neq O$ check is mandatory: the pairing equation is trivially satisfied by (O, O) for any m, so only the explicit guard prevents this degenerate signature from passing.

3 Part II: Chaum-Pedersen Re-Encryption Correctness

3.1 Setup

Alice holds a credential whose ElGamal ciphertext is registered on chain and was verified at registration time against her PS signature:

$$E_{\text{alice}} = (R_a, C_a) = (r_a \cdot G,\ M + r_a \cdot pk_{\text{alice}}), \qquad pk_{\text{alice}} = sk_{\text{alice}} \cdot G.$$

To identify herself to Bob she re-encrypts under pk_{bob} with fresh randomness r_b:

$$E_{\text{bob}} = (R_b, C_b) = (r_b \cdot G,\ M + r_b \cdot pk_{\text{bob}}).$$

She then produces a non-interactive Chaum-Pedersen proof that E_{alice} and E_{bob} encrypt the *same* message point M – without revealing M, m, sk_{alice}, or r_b. The BUCK contract reads E_{alice} from storage (not calldata) and verifies the proof.

3.2 The Statement

Alice proves knowledge of (sk_{alice}, r_b) satisfying the three simultaneous relations

$$pk_{\text{alice}} = sk_{\text{alice}} \cdot G \tag{S1}$$
$$R_b = r_b \cdot G \tag{S2}$$
$$C_a - sk_{\text{alice}} \cdot R_a = C_b - r_b \cdot pk_{\text{bob}}. \tag{S3}$$

(S1) binds her to the registered public key; (S2) binds r_b to the ciphertext she submits; (S3) states that the ElGamal-decrypt of E_{alice} equals the ElGamal-decrypt of E_{bob} under their respective secret keys.

3.3 Protocol (Fiat-Shamir)

Commit. Prover samples $k_1, k_2 \leftarrow \mathbb{Z}_q$ and computes

$$T_1 = k_1 \cdot G, \qquad T_2 = k_2 \cdot G, \qquad T_3 = k_2 \cdot pk_{\text{bob}} - k_1 \cdot R_a.$$

Challenge.

$$e = H(G,\ pk_{\text{alice}},\ pk_{\text{bob}},\ R_a, C_a, R_b, C_b,\ T_1, T_2, T_3,\ \texttt{msg.sender},\ \texttt{spender},\ \texttt{chainid})\,.$$

Respond. $s_1 = k_1 - e \cdot sk_{\text{alice}} \pmod q$, $\qquad s_2 = k_2 - e \cdot r_b \pmod q$.

Verify. The contract recomputes e from the public inputs and the prover-supplied commitments, then checks

$$s_1 \cdot G + e \cdot pk_{\text{alice}} \stackrel{?}{=} T_1 \tag{V1}$$
$$s_2 \cdot G + e \cdot R_b \stackrel{?}{=} T_2 \tag{V2}$$
$$s_2 \cdot pk_{\text{bob}} - s_1 \cdot R_a + e \cdot (C_b - C_a) \stackrel{?}{=} T_3. \tag{V3}$$

3.4 Theorem 3 (Completeness)

Statement. An honestly generated transcript satisfies (V1), (V2), (V3).

Proof. (V1) and (V2) are standard Schnorr identities:

$$s_1 G + e \cdot pk_{\text{alice}} = (k_1 - e \cdot sk_{\text{alice}})G + e \cdot sk_{\text{alice}} G = k_1 G = T_1,$$
$$s_2 G + e \cdot R_b = (k_2 - e r_b)G + e r_b G = k_2 G = T_2.$$

For (V3), substitute s_1, s_2 and rearrange:

$$\begin{aligned} s_2 \cdot pk_{\text{bob}} - s_1 \cdot R_a + e(C_b - C_a) &= (k_2 - e r_b) pk_{\text{bob}} - (k_1 - e \cdot sk_{\text{alice}}) R_a + e(C_b - C_a) \\ &= \big[k_2 pk_{\text{bob}} - k_1 R_a\big] + e\big[sk_{\text{alice}} R_a - r_b \cdot pk_{\text{bob}} + C_b - C_a\big] \\ &= T_3 + e\big[-(C_a - sk_{\text{alice}} R_a) + (C_b - r_b \cdot pk_{\text{bob}})\big] \\ &= T_3 + e \cdot \big[-M + M\big] \\ &= T_3. \end{aligned}$$

where the penultimate line uses the witness relations from the honest setup, and the final line uses (S3). ■

3.5 Theorem 4 (Special Soundness / 2-Extractability)

Statement. Given two accepting transcripts $(T_1, T_2, T_3,\ e,\ s_1, s_2)$ and $(T_1, T_2, T_3,\ e',\ s_1', s_2')$ with $e \neq e'$ (and identical public inputs), there is an efficient algorithm that outputs a witness (sk^*, r^*) satisfying (S1), (S2), (S3).

Proof (extractor). Subtract (V1) for the two transcripts:

$$(s_1 - s_1')G + (e - e') \cdot pk_{\text{alice}} = O \quad \Longrightarrow \quad pk_{\text{alice}} = \frac{s_1' - s_1}{e - e'} \cdot G.$$

Set $sk^* := (s_1' - s_1)(e - e')^{-1} \bmod q$; this is well-defined because q is prime and $e \neq e'$. By construction (S1) holds. Similarly from (V2), $r^* := (s_2' - s_2)(e - e')^{-1} \bmod q$ satisfies (S2): $R_b = r^* G$.

For (S3), subtract (V3) for the two transcripts:

$$(s_2 - s_2') \cdot pk_{\text{bob}} - (s_1 - s_1') \cdot R_a + (e - e')(C_b - C_a) = O.$$

Substitute $s_2 - s_2' = -(e - e')r^*$ and $s_1 - s_1' = -(e - e')sk^*$:

$$-(e - e')r^* \cdot pk_{\text{bob}} + (e - e')sk^* \cdot R_a + (e - e')(C_b - C_a) = O.$$

Dividing through by the nonzero scalar $(e - e')$:

$$sk^* \cdot R_a - r^* \cdot pk_{\text{bob}} + (C_b - C_a) = O \qquad \Longleftrightarrow \qquad C_a - sk^* \cdot R_a = C_b - r^* \cdot pk_{\text{bob}},$$

which is (S3). Therefore (sk^*, r^*) is a valid witness, and the extractor runs in the time of four field inversions and a constant number of group operations. ■

Remark (implication for cheating provers). Special soundness lifts, via the forking lemma[2], to a reduction from any efficient prover that convinces the verifier on a false statement to an efficient solver for discrete logarithms on G_1. Under the DLog assumption on alt_bn128, no such prover exists. In particular, Alice cannot produce a valid proof for E_{bob} encrypting any $M' \neq M$ – not merely "hard to compute," but infeasible under a standard assumption Ethereum already requires.

3.6 Theorem 5 (Honest-Verifier Zero-Knowledge)

Statement. There is an efficient simulator $\mathcal{S}$ that, given only the public inputs and a challenge e drawn from the verifier's distribution, outputs a transcript $(T_1, T_2, T_3, e, s_1, s_2)$ distributed *identically* to an honest prover's transcript conditioned on that challenge.

Proof (simulator). $\mathcal{S}$ samples $s_1, s_2 \leftarrow \mathbb{Z}_q$ uniformly, then *defines*

$$
\begin{aligned}
T_1 &:= s_1 \cdot G + e \cdot pk_{\text{alice}}, \\
T_2 &:= s_2 \cdot G + e \cdot R_b, \\
T_3 &:= s_2 \cdot pk_{\text{bob}} - s_1 \cdot R_a + e \cdot (C_b - C_a).
\end{aligned}
$$

Equations (V1), (V2), (V3) hold by construction, so the transcript is accepting.

Distribution match. In the real protocol, k_1, k_2 are uniform on $\mathbb{Z}_q$; the honest responses $s_i = k_i - e \cdot w_i$ are therefore uniform on $\mathbb{Z}_q$ (shift by a constant). The commitments T_1, T_2, T_3 are *deterministic functions* of the s_i and the public inputs: (V1), (V2), (V3) each uniquely determine one T_i from the corresponding equation. The simulator samples s_i uniformly and computes the T_i via the same equations – the joint distribution over $(T_1, T_2, T_3, s_1, s_2)$ is identical. ■

Corollary (NIZK via Fiat-Shamir in the ROM). Model H as a programmable random oracle. The Fiat-Shamir transform replaces the verifier's challenge with $e = H(\text{public inputs}, T_1, T_2, T_3, \text{context})$. Standard arguments[3] promote:

- HVZK $\Rightarrow$ NIZK (the simulator programs H so that $H(\text{public inputs}, T_1, T_2, T_3, \ldots) = e$ for the (T_i) it produced);
- special soundness $\Rightarrow$ proof-of-knowledge soundness (the forking lemma rewinds the oracle).

The binding of `msg.sender`, `spender`, `chainid` into the hash input is standard domain separation that prevents cross-context replay.

3.7 Sanity Check (py_ecc on BN254)

```
# Build registered credential E_alice for Alice
sk_alice = rand()
pk_alice = multiply(G1, sk_alice)
r_a      = rand()
R_a      = multiply(G1, r_a)
```


[2]Pointcheval, D. and Stern, J. "Security Arguments for Digital Signatures and Blind Signatures." Journal of Cryptology 13(3),

1. The forking lemma: an adversary that convinces a sigma-protocol

verifier with non-negligible probability can be rewound to produce two accepting transcripts with distinct challenges, from which the witness is extracted.

[3]Fiat, A. and Shamir, A. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems." CRYPTO 1986. The transform replacing an interactive verifier's challenge with a hash of the transcript, analyzed in the random oracle model of Bellare and Rogaway (CCS 1993).

```
M        = multiply(G1, m)                          # identity point M = m * G
C_a      = add(M, multiply(pk_alice, r_a))

# Bob's identity public key (Alice knows this from the registry)
sk_bob = rand()
pk_bob = multiply(G1, sk_bob)

# Re-encryption for Bob: E_bob = (r_b * G, M + r_b * pk_bob)
r_b = rand()
R_b = multiply(G1, r_b)
C_b = add(M, multiply(pk_bob, r_b))

def cp_prove(k1, k2):
    T1 = multiply(G1, k1)
    T2 = multiply(G1, k2)
    T3 = add(multiply(pk_bob, k2), neg(multiply(R_a, k1)))
    tag = "|".join(str(v) for v in
                   [G1, pk_alice, pk_bob, R_a, C_a, R_b, C_b, T1, T2, T3])
    e   = int(hashlib.sha256(tag.encode()).hexdigest(), 16) % ORDER
    s1  = (k1 - e * sk_alice) % ORDER
    s2  = (k2 - e * r_b)      % ORDER
    return (T1, T2, T3, e, s1, s2)

def cp_verify(T1, T2, T3, e, s1, s2):
    v1 = eq(add(multiply(G1, s1),    multiply(pk_alice, e)), T1)
    v2 = eq(add(multiply(G1, s2),    multiply(R_b, e)),      T2)
    Cdiff = add(C_b, neg(C_a))
    v3 = eq(add(add(multiply(pk_bob, s2), neg(multiply(R_a, s1))),
                multiply(Cdiff, e)), T3)
    return v1, v2, v3

# --- Theorem 3: completeness ---
proof = cp_prove(rand(), rand())
v1, v2, v3 = cp_verify(*proof)
assert v1 and v2 and v3, "Theorem 3 FAILED"
print("Theorem 3 (completeness):           PASS")

# --- Theorem 4: extract witness from two transcripts with same (T1,T2,T3) ---
# We fix (k1, k2) to produce the same (T1, T2, T3), then force two
# different challenges by hashing with two different context strings
# (in production the different challenges arise from rewinding the RO).
k1, k2 = rand(), rand()
T1 = multiply(G1, k1)
T2 = multiply(G1, k2)
T3 = add(multiply(pk_bob, k2), neg(multiply(R_a, k1)))

def responses(e):
    return (k1 - e * sk_alice) % ORDER, (k2 - e * r_b) % ORDER

e_1 = rand()
e_2 = rand()
while e_1 == e_2:
    e_2 = rand()
s1_a, s2_a = responses(e_1)
s1_b, s2_b = responses(e_2)

def inv_mod(a, p):
    return pow(a, -1, p)

delta_e     = (e_1 - e_2) % ORDER
delta_e_inv = inv_mod(delta_e, ORDER)
sk_star     = ((s1_b - s1_a) * delta_e_inv) % ORDER   # should equal sk_alice
r_star      = ((s2_b - s2_a) * delta_e_inv) % ORDER   # should equal r_b

assert sk_star == sk_alice, "Theorem 4 sk extraction FAILED"
assert r_star  == r_b,      "Theorem 4 r  extraction FAILED"
# Witnesses satisfy (S1), (S2), (S3):
assert eq(multiply(G1, sk_star), pk_alice)
```

```
assert eq(multiply(G1, r_star),  R_b)
assert eq(add(C_a, neg(multiply(R_a, sk_star))),
          add(C_b, neg(multiply(pk_bob, r_star))))
print("Theorem 4 (special soundness):       PASS  (sk, r extracted from two transcripts)")

# --- Theorem 5: simulator produces accepting transcript without witness ---
def cp_simulate(e):
    s1 = rand(); s2 = rand()
    T1 = add(multiply(G1, s1),    multiply(pk_alice, e))
    T2 = add(multiply(G1, s2),    multiply(R_b, e))
    Cdiff = add(C_b, neg(C_a))
    T3 = add(add(multiply(pk_bob, s2), neg(multiply(R_a, s1))),
             multiply(Cdiff, e))
    return (T1, T2, T3, e, s1, s2)

sim = cp_simulate(rand())
v1, v2, v3 = cp_verify(*sim)
assert v1 and v2 and v3, "Theorem 5 FAILED (simulated transcript rejected)"
print("Theorem 5 (HVZK):                    PASS  (simulator accepts without witness)")

# --- Unsoundness counter-example: wrong M at destination ---
M_fake = multiply(G1, rand())
C_b_fake = add(M_fake, multiply(pk_bob, r_b))
# Honest prover tries to produce a proof for (E_alice, (R_b, C_b_fake))
k1, k2 = rand(), rand()
T1 = multiply(G1, k1)
T2 = multiply(G1, k2)
T3 = add(multiply(pk_bob, k2), neg(multiply(R_a, k1)))
tag = "|".join(str(v) for v in
               [G1, pk_alice, pk_bob, R_a, C_a, R_b, C_b_fake, T1, T2, T3])
e_f = int(hashlib.sha256(tag.encode()).hexdigest(), 16) % ORDER
s1_f = (k1 - e_f * sk_alice) % ORDER
s2_f = (k2 - e_f * r_b)      % ORDER
v1_f = eq(add(multiply(G1, s1_f), multiply(pk_alice, e_f)), T1)
v2_f = eq(add(multiply(G1, s2_f), multiply(R_b, e_f)),      T2)
Cdiff_f = add(C_b_fake, neg(C_a))
v3_f = eq(add(add(multiply(pk_bob, s2_f), neg(multiply(R_a, s1_f))),
              multiply(Cdiff_f, e_f)), T3)
assert v1_f and v2_f and not v3_f, "Unsoundness: fake M proof should fail at V3"
print("Counter-example (M != M'):           V1,V2 PASS; V3 FAIL (as required)")


Theorem 3 (completeness):            PASS
Theorem 4 (special soundness):       PASS  (sk, r extracted from two transcripts)
Theorem 5 (HVZK):                    PASS  (simulator accepts without witness)
Counter-example (M != M'):           V1,V2 PASS; V3 FAIL (as required)
```


How to read the output: Four PASS lines plus the counter-example. The counter-example is the mirror image of Theorem 4: when the statement is *false* (different message points), the first two verification equations still pass (Alice's key and randomness are genuine) but (V3) – the "same-message" equation – fails. This is precisely the mathematical impossibility the prose in the Identity document refers to when it says "no valid proof exists."

4 Part III: End-to-End Re-Encryption Correctness

4.1 Chain of Trust

The on-chain identity pipeline chains together the two primitives proved above:

```
KYC ceremony:   issuer signs m      -->  sigma on m                 (q-SDH)
Fountain:       rerandomize sigma   -->  sigma' on same m           (Theorem 1, 2)
Registration:   PS + NIZK check     -->  E_alice bound to same m    (PS verify + NIZK soundness)
approve(Bob):   Alice re-encrypts   -->  E_bob                      (Theorem 3)
                Chaum-Pedersen      -->  verified on chain          (Theorems 3, 4, 5)
Decryption:     Bob computes M      -->  M = C_b - sk_bob * R_b     (ElGamal correctness)
Two-channel:    Bob receives        -->  H(identity_data) * G == M  (hash commitment)
```

4.2 Theorem 6 (End-to-End Re-Encryption Correctness)

Statement. Suppose the BUCK contract:

1. holds a registered credential $E_{\text{alice}} = (R_a, C_a)$ together with pk_{alice} and a PS signature σ' that verified at registration time against a trusted issuer public key (X, Y);

2. accepts a Chaum-Pedersen proof π supplied by `msg.sender` = Alice, with spender pk_{bob} and submitted ciphertext $E_{\text{bob}} = (R_b, C_b)$.

Then, under the DLog assumption on G_1 and in the ROM, there exists $M = m \cdot G$ such that

$$C_a - sk_{\text{alice}} \cdot R_a \;=\; M \;=\; C_b - sk_{\text{bob}} \cdot R_b,$$

and the scalar m is the issuer-certified identity scalar covered by σ'.

Proof sketch. The registration check evaluates $e(\sigma'_1, X + mY) = e(\sigma'_2, g_2)$ together with a NIZK that binds the same m to E_{alice} (so $C_a - sk_{\text{alice}} R_a = m \cdot G$); existential unforgeability of PS under q-SDH ensures m is the scalar the issuer signed. The contract reads E_{alice} from storage, not calldata, so the value is frozen.

Theorem 4 (special soundness) plus the forking lemma reduction to DLog gives: from any adversary producing an accepting π with non-negligible probability there is an extractor outputting witnesses (sk^*, r^*) with $C_a - sk^* R_a = C_b - r^* pk_{\text{bob}}$. Since $pk_{\text{alice}} = sk^* \cdot G = sk_{\text{alice}} \cdot G$ and G is a generator, $sk^* = sk_{\text{alice}}$; so both sides equal the registered M. Therefore E_{bob} decrypts under sk_{bob} to the same M that the issuer's PS signature attests to. ■

4.3 Why This Suffices for Every Downstream Use Case

- `approve` / `transfer` **(Identity):** Bob decrypts E_{bob} and obtains M; combined with off-chain `identity_data` delivery and the check $H(\text{identity\_data}) \cdot G = M$, he learns Alice's real identity with cryptographic assurance.

- **Wormhole SNARK (Wormhole):** The source-to-destination binding is the same Chaum-Pedersen statement, lifted inside a SNARK. Theorem 4's soundness lifts directly: a SNARK-verified instance of the Chaum-Pedersen relation implies the same extractable witness relation, so the burn and mint credentials share M. Identity laundering through the wormhole is therefore infeasible.

- **Identity Guardians (`approveFor`, recovery, cosigning):** The Schnorr proof of knowledge of sk is a one-statement specialization of the above (only (S1) with a Fiat-Shamir binding to (`msg.sender`, `principal`)). Theorems 3-5 apply verbatim with the second and third statements dropped.

5 Security Assumptions Summary

Theorem	Primitive	Assumption needed	Conditional on crypto?
Theorem 1 (PS completeness)	PS	bilinearity of e	**no** (unconditional)
Theorem 2 (PS unlinkability)	PS	prime-order G_1, uniform t	**no** (information-theoretic)
Theorem 3 (CP completeness)	CP	group axioms	**no** (unconditional)
Theorem 4 (CP special soundness)	CP	none for extractor; DLog for cheating-prover reduction	DLog (only for reduction)
Theorem 5 (CP HVZK)	CP	uniform k_1, k_2	**no** (perfect simulation)
NIZK corollary	CP + FS	ROM	ROM
Theorem 6 (end-to-end)	PS + CP	q-SDH (PS unforgeability), DLog, ROM	yes

Every assumption on the "yes" rows is already load-bearing for Ethereum itself (ECDSA on secp256k1 assumes DLog; every pairing-based SNARK verifier on L1 uses the same ROM reductions; every existing BLS-based rollup assumes q-SDH-class primitives). Alberta Buck introduces *no novel* cryptographic assumption.

6 Implementation Notes

6.1 What the Contract Must Enforce

The theorems assume a well-formed verifier. Three conditions on the on-chain code are load-bearing:

1. **Registration: $\sigma'_1 \neq O$.** Required by Theorem 1's non-degeneracy argument. A missing check admits the trivial signature attack (see Identity doc, Attack 9 / Link 1).

2. **Registration: G_2 subgroup check.** The `ecPairing` precompile (post-EIP-197) validates that X, Y lie in the correct prime-order subgroup of G_2. Without this, small-subgroup attacks on the twist curve could forge pairings. Theorem 1 is stated over the correct subgroup; enforcement is the precompile's responsibility.

3. **Verification: E_{alice} read from storage, not calldata.** Theorem 6's reduction routes through the registration-time binding of E_{alice} to issuer-certified m. If Alice supplies E_{alice} via calldata the binding is severed and Theorem 6 does not apply.

6.2 What the Prover Must Enforce

- **Uniform randomness for t, k_1, k_2, r_b.** Theorems 2 and 5 depend on uniform distributions. A biased RNG converts statistical unlinkability into a detectable correlation and converts perfect HVZK into approximate HVZK with an adversarial advantage proportional to the bias.

- **Fiat-Shamir domain separation.** The hash input must include all public inputs and context (`msg.sender`, `spender`, `chainid`). Omitting any one opens a cross-context replay path; the forking lemma reduction is unaffected, but the NIZK is no longer bound to the intended transaction.

6.3 Scope Not Covered Here

- *Knowledge soundness* of the registration-time NIZK that binds σ' to E_{alice}: this is a standard Schnorr-family sigma protocol over a linear pairing relation (see `alberta-buck-identity.org`, "One-Time (PS Signature + NIZK Verification at Registration)") and follows the same completeness / soundness / ZK template as Theorems 3-5. A separate write-up at the same level of detail is the natural next step.

- *Circuit soundness* of the wormhole SNARK: the cryptographic statement inside the SNARK is the same Chaum-Pedersen relation, so Theorem 4 applies to the *statement* being proved. Soundness of the *proof system* (Groth16 / PLONK / STARK) is assumed separately and depends on the chosen proof system's trusted setup (if any) and circuit correctness; this is orthogonal to the contents of this document.