Sharp: Short Relaxed Range Proofs

1 Motivation

Range proofs enable a prover to convince a verifier that a committed or encrypted integer lies within an interval without revealing its value, which are indispensable in privacy-preserving cryptosystems, There are currently two main types of range proof:

- **n-ary Decomposition (e.g. Bulletproofs):** These protocols express each value in a radix-n representation and prove via inner-product arguments that each digit lies in $[0, n-1]$. Using logarithmic-sized proofs and standard elliptic curves, Bulletproofs achieve proof sizes of $O(\lambda + \log N)$ group elements for N values, with verification cost also logarithmic in N. However, they require specialized vector commitments and multi-exponentiation techniques.
- **Square Decomposition (e.g. CKLR):** Building on Lagrange's four-square theorems, CKLR decomposes a quadratic expression $4x(B-x)+1$ into a sum of four squares to enforce $x \in [0, B]$. This yields a proof size linear in the number of decomposed squares but avoids complex vector commitments, relying only on standard Pedersen commitments over elliptic curves. The proof overhead is $O(\lambda N)$ group elements, and batching requires nontrivial extensions (which is the main highlight of the Sharp research paper).

While n-ary proofs minimize asymptotic size, they incur higher constant factors and rely on vector-commitment primitives. $\mathsf{Sharp}_{\mathsf{GS}}$ is part of the Sharp family of range-proof protocols, designed to optimize both proof compactness and efficiency. Square-based proofs offer simpler setups but larger per-value overhead. $\mathsf{Sharp}_{\mathsf{GS}}$ uses:

- A **three-square polynomial test** reducing per-value commitments to 3 elements, cutting overhead from CKLR's four-square approach.
- Log-size batching via **Pedersen multi-commitments** and group switching
- **Group-switching** enables use of a standard 256-bit elliptic curve for main commitments while switching to a suitably sized curve for the decomposition proof, via transparent common reference string generation.

2 Three Squares Decomposition Algorithm

The core innovation of $\mathsf{Sharp}_{\mathsf{GS}}$ relies on efficiently computing three-square decompositions to prove range membership. This section details the mathematical foundations and algorithmic approach.

2.1 Legendre's Three-Square Theorem

Legendre's three-square theorem states that a positive integer n can be expressed as a sum of three squares iff n is not of the form $4^a(8b+7)$ where $a, b \geq 0$ are integers.

For range proofs, we use the expression:

$$4x(B-x)+1 = y_1^2 + y_2^2 + y_3^2 \tag{1}$$

When $x \in [0, B]$, we have $4x(B-x) \geq 0$, which means $4x(B-x)+1 \geq 1$.

2.2 Algorithm

Before attempting decomposition, we verify that the target number $n = 4x(B-x)+1$ satisfies Legendre's criterion:

$$\left\lfloor \frac{n}{4^{\mathrm{val}_4(n)}} \right\rfloor \bmod 8 \neq 7 \tag{2}$$

where $\mathrm{val}_4(n)$ is the largest power of 4 dividing n.

The main algorithm proceeds as follows. For each candidate $z \in [0, \lfloor\sqrt{n}\rfloor]$, we compute:

$$m = n - z^2 \tag{3}$$

In practice, empirical testing shows that for numbers up to 1024 bits, all required z values are found within the much smaller range $[0, 10^6]$, making the algorithm highly efficient even for large inputs.

We then determine if m can be written as $x^2 + y^2$ using the sum-of-two-squares theorem. An integer $m > 0$ can be expressed as a sum of two squares iff every prime $p \equiv 3 \pmod 4$ appears to an even power in the prime factorization of m.

When m satisfies this criterion, we find the actual decomposition using algebraic number theory over $\mathbb{Z}[i]$ (the Gaussian integers). Specifically, we find $\alpha = a + bi \in \mathbb{Z}[i]$ such that $N(\alpha) = a^2 + b^2 = m$.

2.3 Implementation via PARI/GP

The decomposition is computed using PARI/GP's built-in algebraic number theory functions.

A *Gaussian integer* is a complex number of the form $a + bi$ where $a, b \in \mathbb{Z}$ are integers and $i^2 = -1$. The set of all Gaussian integers forms a ring denoted $\mathbb{Z}[i] = \{a + bi : a, b \in \mathbb{Z}\}$.

The *norm* of a Gaussian integer $\alpha = a + bi$ is defined as:

$$N(\alpha) = N(a + bi) = a^2 + b^2 \tag{4}$$

The key insight is that finding integers a, b such that $a^2+b^2 = m$ is equivalent to finding a Gaussian integer $\alpha = a+bi$ whose norm equals m. This transforms the sum-of-squares problem into a norm equation in the Gaussian integers. The PARI/GP function `bnfinit(x^2 + 1)` initializes the number field $\mathbb{Q}(i)$ and its ring of integers $\mathbb{Z}[i]$. The polynomial $x^2 + 1$ defines the minimal polynomial of i over $\mathbb{Q}$, establishing the algebraic structure where $i^2 = -1$. Subsequently, `bnfisintnorm(K, m)` efficiently finds all Gaussian integers $\alpha \in \mathbb{Z}[i]$ with $N(\alpha) = m$ by solving the norm equation in the ring. The algorithm returns these elements as polynomials in i, from which we extract the integer coefficients a and b to obtain the desired decomposition $m = a^2 + b^2$.

2.4 Connection to $\mathsf{Sharp}_{\mathsf{GS}}$

The three-square decomposition enables the range proof by establishing:

$$x \in [0, B] \iff 4x(B - x) + 1 \text{ admits a three-square representation} \tag{5}$$

This reduces the range-checking problem to verifying polynomial relationships between the committed values and their square decompositions, which can be done efficiently using the polynomial test in $\mathsf{Sharp}_{\mathsf{GS}}$.

3 Setup and Parameters

Goal: Prove that N committed values $x_1, \ldots, x_N$ each belong to range $[0, B]$

3.1 Protocol Parameters

- N: number of values to prove
- R: number of repetitions for security (typically $R = \lceil \lambda / \log(\Gamma + 1) \rceil$)
- Γ: challenge space size $[0, \Gamma]$
- S: randomness space size $[0, S]$ (hiding parameter)
- L_x, L_r: masking overheads
- $\mathcal{R}_x, \mathcal{R}_r$: masking distributions
These distributions sample masking randomness vectors of size L_x and L_r, respectively, drawn uniformly from their domains (x from $[0, p-1]$, r from $[0, q-1]$)
- $\mathsf{p}_x, \mathsf{p}_r$: masking abort probabilities
- Hash function: $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^{2\lambda}$

3.2 System Setup

The protocol requires a Common Reference String (CRS) that can be generated transparently as follows:

1. Generate two cryptographic groups:
   - $\mathbb{G}_{\mathsf{com}}$ (for main value commitments) with prime order $p > 2(B\Gamma^2 + 1)L_x$
   - $\mathbb{G}_{\mathsf{3sq}}$ (for decomposition proof) with prime order $q > 18((B\Gamma + 1)L_x)^2$

   Group switching optimizes efficiency by choosing appropriate group sizes for each purpose.
2. Sample generators using transparent methods:
$$G_0, G_1, \ldots, G_N, G_{1,1}, G_{1,2}, G_{1,3}, \ldots, G_{N,1}, G_{N,2}, G_{N,3} \xleftarrow{\$} \mathbb{G}_{\mathsf{com}} \tag{6}$$
$$H_0, H_1, \ldots, H_N \xleftarrow{\$} \mathbb{G}_{\mathsf{3sq}} \tag{7}$$
3. Define commitment keys as:
$$\mathsf{ck}_{\mathbb{G}_{\mathsf{com}}} = \left(G_0, \{G_i\}_{i=1}^N, \{G_{i,j}\}_{i\in[1,N], j\in[1,3]}\right) \tag{8}$$
$$\mathsf{ck}_{\mathbb{G}_{\mathsf{3sq}}} = (H_0, \{H_i\}_{i=1}^N) \tag{9}$$
4. Define CRS as:
$$\mathsf{crs} = (\mathsf{ck}_{\mathbb{G}_{\mathsf{com}}}, \mathsf{ck}_{\mathbb{G}_{\mathsf{3sq}}}) \tag{10}$$

4 $\mathsf{Sharp}_{\mathsf{GS}}$ Algorithm

Input:

- **Both parties**: CRS $\mathsf{crs} = (\mathsf{ck}_{\mathbb{G}_{\mathsf{com}}}, \mathsf{ck}_{\mathbb{G}_{\mathsf{3sq}}})$, statement $C_x = r_x G_0 + \sum_{i=1}^N x_i G_i$ and range bound B
- **Prover**: Witnesses $(x_1, \ldots, x_N) \in [0, B]^N$ and randomness $r_x \in [0, S]$

4.1 Phase 1: Prover's First Message

4.1.1 Compute Three-Square Decomposition:

For each $i \in [1, N]$:

$$4x_i(B - x_i) + 1 = \sum_{j=1}^{3} y_{i,j}^2 \tag{11}$$

4.1.2 Commit to Decomposition:

$$C_y = r_y G_0 + \sum_{i=1}^{N} \sum_{j=1}^{3} y_{i,j} G_{i,j} \tag{12}$$

where $r_y \xleftarrow{\$} [0, S]$.

4.1.3 For each repetition $k \in [1, R]$:

a) Sample Random Masks:

- Opening masks: $\tilde{r}_{k,x}, \tilde{r}_{k,y} \xleftarrow{\$} \mathcal{R}_r$
- Value masks: $\tilde{x}_{k,i} \xleftarrow{\$} \mathcal{R}_x$ for $i \in [1, N]$
- Decomposition masks: $\tilde{y}_{k,i,j} \xleftarrow{\$} \mathcal{R}_x$ for $i \in [1, N], j \in [1, 3]$

b) Create Masked Commitments:

$$D_{k,x} = \tilde{r}_{k,x} G_0 + \sum_{i=1}^{N} \tilde{x}_{k,i} G_i \tag{13}$$

$$D_{k,y} = \tilde{r}_{k,y} G_0 + \sum_{i=1}^{N} \sum_{j=1}^{3} \tilde{y}_{k,i,j} G_{i,j} \tag{14}$$

c) Prepare Polynomial Coefficients:

For $i \in [1, N]$, the prover commits to coefficients $\alpha^*_{1,k,i}$ and $\alpha^*_{0,k,i}$ such that when the verifier later computes $f^*_{k,i} = 4z_{k,i}(\gamma_k B - z_{k,i}) + \gamma_k^2 - \sum_{j=1}^3 z_{k,i,j}^2$, it will equal $\alpha^*_{1,k,i}\gamma_k + \alpha^*_{0,k,i}$ (degree 1 in γ_k) iff the three-square decomposition holds.

$$\alpha^*_{1,k,i} = 4\tilde{x}_{k,i} B - 8x_i \tilde{x}_{k,i} - 2\sum_{j=1}^{3} y_{i,j} \tilde{y}_{k,i,j} \quad \text{(coefficient of } \gamma_k) \tag{15}$$

$$\alpha^*_{0,k,i} = -\left(4\tilde{x}_{k,i}^2 + \sum_{j=1}^{3} \tilde{y}_{k,i,j}^2\right) \quad \text{(constant term)} \tag{16}$$

Commit to these:

$$C_{k,*} = r_k^* H_0 + \sum_{i=1}^{N} \alpha_{1,k,i}^* H_i \tag{17}$$

$$D_{k,*} = \tilde{r}_k^* H_0 + \sum_{i=1}^{N} \alpha_{0,k,i}^* H_i \tag{18}$$

where $r_k^* \xleftarrow{\$} [0, S]$ and $\tilde{r}_k^* \xleftarrow{\$} \mathcal{R}_r$.

Send $C_y, \{C_{k,*}, D_{k,x}, D_{k,y}, D_{k,*}\}_{k=1}^R$ to verifier.

4.2 Phase 2: Verifier's Challenge

Samples challenges $\gamma_k \xleftarrow{\$} [0, \Gamma]$ for each repetition $k \in [1, R]$. **Send** $\{\gamma_k\}_{k=1}^R$ to prover.

4.3 Phase 3: Prover's Response

For each repetition $k \in [1, R]$:

4.3.1 Mask the Witnesses

$$z_{k,i} = \mathsf{mask}_x(\gamma_k \cdot x_i, \tilde{x}_{k,i}) \quad \text{(masked value)} \tag{19}$$

$$z_{k,i,j} = \mathsf{mask}_x(\gamma_k \cdot y_{i,j}, \tilde{y}_{k,i,j}) \quad \text{(masked decomposition)} \tag{20}$$

where $\mathsf{mask}_x(v, r)$ outputs $v + r$ if $v + r \in [0, (B\Gamma + 1)L_x]$, else $\perp$.

4.3.2 Mask the Randomness

$$t_{k,x} = \mathsf{mask}_r(\gamma_k r_x, \tilde{r}_{k,x}) \tag{21}$$

$$t_{k,y} = \mathsf{mask}_r(\gamma_k \cdot r_y, \tilde{r}_{k,y}) \tag{22}$$

$$t_k^* = \mathsf{mask}_r(\gamma_k \cdot r_k^*, \tilde{r}_k^*) \tag{23}$$

4.3.3 Abort Handling

If any mask ($z_{k,i}$, $z_{k,i,j}$, $t_{k,x}$, $t_{k,y}$, t_k^*) returns $\perp$, the prover must restart the masking for that repetition k:

1. Discard all masks $\{z_{k,i},\, z_{k,i,j},\, t_{k,x},\, t_{k,y},\, t_k^*\}$.
2. Resample new randomness $\{\tilde{x}_{k,i}, \tilde{y}_{k,i,j}, \tilde{r}_{k,x}, \tilde{r}_{k,y}, \tilde{r}_k^*\}$.
3. Recompute masked values via mask_x and mask_r.
4. Repeat the abort check.

This rejection-sampling ensures both the hiding property and the numeric bounds.

Send $\{z_{k,i,j}, z_{k,i}, t_{k,x}, t_{k,y}, t_k^*\}_{k \in [1,R], i \in [1,N], j \in [1,3]}$ to verifier.

4.4 Phase 4: Verifier's Verification

For each repetition $k \in [1, R]$:

4.4.1 Check 1: Commitment Consistency

Verify:

$$D_{k,x} + \gamma_k C_x \stackrel{?}{=} t_{k,x} G_0 + \sum_{i=1}^{N} z_{k,i} G_i \tag{24}$$

$$D_{k,y} + \gamma_k C_y \stackrel{?}{=} t_{k,y} G_0 + \sum_{i=1}^{N} \sum_{j=1}^{3} z_{k,i,j} G_{i,j} \tag{25}$$

4.4.2 Check 2: Polynomial Degree

Compute:

$$f^*_{k,i} = 4 z_{k,i}(\gamma_k B - z_{k,i}) + \gamma_k^2 - \sum_{j=1}^{3} z_{k,i,j}^2 \tag{26}$$

Verify:

$$D_{k,*} + \gamma_k C_{k,*} \stackrel{?}{=} t^*_k H_0 + \sum_{i=1}^{N} f^*_{k,i} H_i \tag{27}$$

If the three-square decomposition holds, then the polynomials $f^*_{k,i}$ should have degree exactly 1 in γ_k. Suppose the difference polynomial is non-zero of degree d, it can only vanish on at most d points of S. By the Schwartz-Zippel lemma,

$$\Pr_{\gamma \leftarrow S}\big[g(\gamma) = 0\big] \;\leq\; \frac{d}{|S|} = \frac{d}{\Gamma + 1}$$

which ensures soundness by detecting non-zero difference polynomials with high probability.

For all $i \in [1, N], j \in [1, 3], k \in [1, R]$:

4.4.3 Check 3: Range Verification

Verify:

$$z_{k,i},\; z_{k,i,j} \stackrel{?}{\in} [0,\; (B\Gamma + 1)L_x] \tag{28}$$

Accept iff all checks succeed for all repetitions $k \in [1, R]$.

5 Hash Function Optimizations

The hash function $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ enables two optimizations.

5.1 Communication Optimization

In the basic protocol, Phase 1 requires the prover to send $4R$ group elements

$$\mathcal{D} = \{C_{k,*}, D_{k,x}, D_{k,y}, D_{k,*}\}_{k=1}^{R} \tag{29}$$

The prover can replace the commitment transmission with a compact hash through the following process:

1. Compute hash $\Delta \leftarrow \mathsf{Hash}(\mathcal{D})$
2. Send compressed message (C_y, Δ) instead of $(C_y, \mathcal{D})$
3. The verifier recomputes $\mathcal{D}$ during verification and checks $\mathsf{Hash}(\mathcal{D}) \stackrel{?}{=} \Delta$

Communication savings:

$$\text{Original size} \quad 4R \times 32\text{-}48 \text{ bytes} \tag{30}$$

$$\text{Optimized size} \quad \frac{2\lambda}{8} \text{ bytes} = 32 \text{ bytes (for } \lambda = 128) \tag{31}$$

The hash optimization preserves security through collision resistance. It would be impossible for any adversary attempting to forge different commitments $\mathcal{D}' \neq \mathcal{D}$ with the same hash $\mathsf{Hash}(\mathcal{D}') = \mathsf{Hash}(\mathcal{D}) = \Delta$.

5.2 Fiat-Shamir Transformation

The hash function enables conversion to a non-interactive zero-knowledge proof via the Fiat-Shamir transformation.

Instead of receiving challenges from the verifier, the prover computes them deterministically. The hash function is applied to generate a seed, which is then used to derive individual challenges:

$$\text{seed} \leftarrow \mathsf{Hash}(\text{statement} \| C_y \| \Delta \| \text{context}) \tag{32}$$

$$\gamma_k \leftarrow \mathsf{Hash}(\text{seed} \| k) \bmod (\Gamma + 1) \quad \text{for } k \in [1, R] \tag{33}$$

where context includes any additional protocol parameters or public inputs, and $\Delta = \mathsf{Hash}(\mathcal{D})$ when using hash optimization.

The modified protocol proceeds as follows:

1. Prover computes first message $(C_y, \mathcal{D})$
2. Prover generates challenges $\{\gamma_k\}_{k=1}^{R}$ using Hash
3. Prover computes response $\{z_{k,i,j}, z_{k,i}, t_{k,x}, t_{k,y}, t_k^*\}$
4. Prover outputs proof $\pi = (C_y, \Delta, \text{response})$
5. Verifier recomputes challenges using Hash and verifies.

The Fiat-Shamir transformation is provably secure in the Random Oracle Model (ROM), where Hash is modeled as a truly random function. The transformation preserves the soundness and zero-knowledge properties of the original interactive protocol.

6 Cost Analysis

The computational complexity of $\mathsf{Sharp}_{\mathsf{GS}}$ can be expressed as:

$$\mathcal{O}(R \cdot N)_{\mathbb{G}_{\mathsf{com}}} + \mathcal{O}(R \cdot N)_{\mathbb{G}_{\mathsf{3sq}}} + \mathcal{O}(R \cdot N)_{\mathbb{F}_p} + \mathcal{O}(R \cdot N)_{\mathbb{F}_q}$$

where:

- $\mathbb{G}_{\mathsf{com}}$ represents group operations in the commitment group (256-333 bits)
- $\mathbb{G}_{\mathsf{3sq}}$ represents group operations in the three-squares group (256-411 bits)
- $\mathbb{F}_p$ and $\mathbb{F}_q$ represent field operations in their respective finite fields

7 Security Guarantees

Let p be the prime order of our group and let N, D be positive integers satisfying

$$N \cdot D < \frac{p}{2}.$$

We define

$$\mathbb{Q}_{N,D} \;=\; \left\{ \tfrac{n}{d} \in \mathbb{Q} \;\middle|\; |n| \leq N,\ 1 \leq d \leq D \right\}.$$

Then for any $x \in \mathbb{Z}_p$ which admits a representative in $\mathbb{Q}_{N,D}$, that representative is unique. This unique fraction is denoted by

$$[x]_{\mathbb{Q}} \;=\; \frac{n}{d} \;\in\; \mathbb{Q}_{N,D}, \quad \text{characterized by} \quad n \;\equiv\; x \cdot d \pmod{p}.$$

$\mathsf{Sharp}_{\mathsf{GS}}$ proves that each x_i has a rational representative $[x_i]_{\mathbb{Q}}$ in $\left[-\frac{1}{4}B, B + \frac{1}{4}B\right]_{\mathbb{Q}}$ with numerator bounded by $K = (B\Gamma + 1)L_x$ and denominator bounded by Γ. This relaxed binding enables several optimizations that would be impossible with integer binding.

7.1 Group Switching Optimization

By accepting rational representatives, $\mathsf{Sharp}_{\mathsf{GS}}$ can work over finite field groups $\mathbb{G}_{\mathsf{com}}$ and $\mathbb{G}_{\mathsf{3sq}}$ instead of hidden order groups. This allows independent optimization of group sizes for commitments (256-333 bits) versus decomposition proofs (256-411 bits).

7.2 Polynomial Verification Technique

The three-square decomposition verification in Phase 4 uses polynomial relationships over finite fields rather than exact arithmetic in hidden order groups, which works efficiently over $\mathbb{Z}_p$ when only rational binding is required.

7.3 Efficient Batching

Pedersen multi-commitments $C_y = r_y G_0 + \sum_{i,j} y_{i,j} G_{i,j}$ work efficiently with rational representatives, enabling batch proofs for N values simultaneously. Integer binding would require separate commitments.

7.4 Simplified Masking Protocol

The masking scheme in Phase 3 operates over finite fields with bounds $z_{k,i} \in [0, (B\Gamma + 1)L_x]$ that correspond directly to rational representative numerator bounds. This enables uniform rejection sampling instead of Gaussian sampling required for integer binding schemes.

8 Other Sharp Algorithm Variants

8.1 $\mathsf{Sharp}_{\mathsf{SO}}^{\mathsf{Po}}$

$\mathsf{Sharp}_{\mathsf{SO}}^{\mathsf{Po}}$ optimizes runtime performance through a fractional shortness test, enabling single-scalar repetitions regardless of batch size. It is designed specifically for standard 256-bit elliptic curves.

8.2 $\mathsf{Sharp}_{\mathsf{RSA}}$

$\mathsf{Sharp}_{\mathsf{RSA}}$ augments the basic construction with RSA groups to achieve standard soundness, binding provers to integer values rather than rationals. Requires trusted RSA parameter generation.

8.3 $\mathsf{Sharp}_{\mathsf{CL}}$

$\mathsf{Sharp}_{\mathsf{CL}}$ uses class groups of imaginary quadratic fields to bind provers to dyadic rationals of form $m/2^k$. Maintains transparent setup.

8.4 $\mathsf{Sharp}_{\mathsf{HO}}$

$\mathsf{Sharp}_{\mathsf{HO}}$ provides a general framework for augmenting any Sharp variant with hidden order groups. Supports RSA groups, class groups, and other hidden order instantiations.