SNARK-friendly primitives (draft)

Daira Hopwood, SNARK-friendly primitives Working Group

June 15, 2020

Abstract

This document was prepared as a starting point for the SNARK-friendly primitives Working Group, which aims to standardize cryptographic primitives for use in SNARKs.

1 Introduction

TODO

2 Notation

$\mathbb{B}$ means the type of bit values, i.e. $\{0, 1\}$. $\mathbb{B}^{\mathbb{Y}}$ means the type of byte values, i.e. $\{0\,..\,255\}$.

$\mathbb{N}$ means the type of nonnegative integers. $\mathbb{N}^+$ means the type of positive integers. $\mathbb{Z}$ means the type of integers.

$x : T$ is used to specify that x has type T. A cartesian product type is denoted by $S \times T$, and a function type by $S \to T$. An argument to a function can determine other argument or result types.

The type of a randomized algorithm is denoted by $S \xrightarrow{\mathrm{R}} T$. The domain of a randomized algorithm may be (), indicating that it requires no arguments. Given $f : S \xrightarrow{\mathrm{R}} T$ and $s : S$, sampling a variable $x : T$ from the output of f applied to s is denoted by $x \xleftarrow{\mathrm{R}} f(s)$.

Initial arguments to a function or randomized algorithm may be written as subscripts, e.g. if $x : X$, $y : Y$, and $f : X \times Y \to Z$, then an invocation of $f(x, y)$ can also be written $f_x(y)$.

$\{x : T \mid p_x\}$ means the subset of x from T for which p_x (a boolean expression depending on x) holds.

$T \subseteq U$ indicates that T is an inclusive subset or subtype of U. $S \cup T$ means the set union of S and T.

$S \cap T$ means the set intersection of S and T, i.e. $\{x : S \mid x \in T\}$.

$S \setminus T$ means the set difference obtained by removing elements in T from S, i.e. $\{x : S \mid x \notin T\}$.

$x : T \mapsto e_x : U$ means the function of type $T \to U$ mapping formal parameter x to e_x (an expression depending on x). The types T and U are always explicit.

$x : T \mapsto_{\neq y} e_x : U$ means $x : T \mapsto e_x : U \cup \{y\}$ restricted to the domain $\{x : T \mid e_x \neq y\}$ and range U.

$\mathscr{P}(T)$ means the powerset of T.

$T^{[\ell]}$, where T is a type and ℓ is an integer, means the type of sequences of length ℓ with elements in T. For example, $\mathbb{B}^{[\ell]}$ means the set of sequences of ℓ bits, and $\mathbb{B}^{\mathbb{Y}[k]}$ means the set of sequences of k bytes.

$\mathbb{B}^{\mathbb{Y}[\mathbb{N}]}$ means the type of byte sequences of arbitrary length.

$\mathsf{length}(S)$ means the length of (number of elements in) S.

`0x` followed by a string of `monospace` hexadecimal digits means the corresponding integer converted from hexadecimal.

"..." means the given string represented as a sequence of bytes in US-ASCII. For example, **"abc"** represents the byte sequence [`0x61`, `0x62`, `0x63`].

$a..b$, used as a subscript, means the sequence of values with indices a through b inclusive. For example, $x_{1..3}$ means the sequence $[x_1, x_2, x_3]$.

$\{a\,..\,b\}$ means the set or type of integers from a through b inclusive.

[$f(x)$ for x from a up to b] means the sequence formed by evaluating f on each integer from a to b inclusive, in ascending order.

Similarly, [$f(x)$ for x from a down to b] means the sequence formed by evaluating f on each integer from a to b inclusive, in descending order.

$a \,||\, b$ means the concatenation of sequences a then b.

$\mathsf{concat}_{\mathbb{B}}(S)$ means the sequence of bits obtained by concatenating the elements of S viewed as bit sequences.

$\mathbb{F}_n$ means the finite field with n elements, and $\mathbb{F}_n^*$ means its group under multiplication (which excludes 0).

Where there is a need to make the distinction, we denote the unique representative of $a : \mathbb{F}_n$ in the range $\{0\,..\,n-1\}$ (or the unique representative of $a : \mathbb{F}_n^*$ in the range $\{1\,..\,n-1\}$) as $a \bmod n$. Conversely, we denote the element of $\mathbb{F}_n$ corresponding to an integer $k : \mathbb{Z}$ as $k \pmod n$. We also use the latter notation in the context of an equality $k = k' \pmod n$ as shorthand for $k \bmod n = k' \bmod n$, and similarly $k \neq k' \pmod n$ as shorthand for $k \bmod n \neq k' \bmod n$. (When referring to constants such as 0 and 1 it is usually not necessary to make the distinction between field elements and their representatives, since the meaning is normally clear from context.)

$\mathbb{F}_n[z]$ means the ring of polynomials over z with coefficients in $\mathbb{F}_n$.

$a + b$ means the sum of a and b. This may refer to addition of integers, finite field elements, or group elements according to context.

$-a$ means the value of the appropriate integer, finite field, or group type such that $(-a) + a = 0$ (or when a is an element of a group $\mathbb{G}$, $(-a) + a = \mathcal{O}_{\mathbb{G}}$), and $a - b$ means $a + (-b)$.

$a \cdot b$ means the product of multiplying a and b. This may refer to multiplication of integers or finite field elements according to context (this notation is not used for group elements).

a/b, also written $\frac{a}{b}$, means the value of the appropriate integer or finite field type such that $(a/b) \cdot b = a$.

$a \bmod q$, for $a : \mathbb{N}$ and $q : \mathbb{N}^+$, means the remainder on dividing a by q. (This usage does not conflict with the notation above for the unique representative of a field element.)

$a \oplus b$ means the bitwise-exclusive-or of a and b, and $a \wedge b$ means the bitwise-and of a and b. These are defined on integers or (equal-length) bit sequences according to context.

$\sum_{i=1}^{n} a_i$ means the sum of $a_{1..n}$. $\prod_{i=1}^{n} a_i$ means the product of $a_{1..n}$. When $N = 0$ these yield the appropriate neutral element, i.e. $\sum_{i=1}^{0} a_i = 0$ and $\prod_{i=1}^{0} a_i = 1$.

$\sqrt[+]{a}$, where $a : \mathbb{F}_q$, means the positive (i.e. in the range $\{0\,..\,\frac{q-1}{2}\}$) square root of a in $\mathbb{F}_q$. It is only used in cases where the square root must exist.

$b\ ?\ x : y$ means x when $b = 1$, or y when $b = 0$.

a^b, for a an integer or finite field element and $b : \mathbb{Z}$, means the result of raising a to the exponent b, i.e.

$$a^b := \begin{cases} \prod_{i=1}^{b} a, & \text{if } b \geq 0 \\ \prod_{i=1}^{-b} \frac{1}{a}, & \text{otherwise.} \end{cases}$$

The $[k]\,P$ notation for scalar multiplication in a group is defined in Section 3.8.

The convention of affixing $\star$ to a variable name is used for variables that denote bit-sequence representations of group elements.

The binary relations $<$, $\leq$, $=$, $\geq$, and $>$ have their conventional meanings on integers, and are defined lexicographically on sequences of integers.

$\mathsf{floor}(x)$ means the largest integer $\leq x$. $\mathsf{ceiling}(x)$ means the smallest integer $\geq x$.

$\mathsf{bitlength}(x)$, for $x : \mathbb{N}$, means the smallest integer ℓ such that $2^\ell > x$.

The symbol $\perp$ is used to indicate unavailable information, or a failed decryption or validity check.

We use the abbreviation "ctEdwards" to refer to complete Twisted Edwards curves and coordinates (see Section 4).

3 Abstract Cryptographic Schemes

3.1 Hash Functions

3.1.1 Collision-Resistant Hash Functions

TODO

3.1.2 General Hash Functions

TODO

3.1.3 Sponges

TODO

3.2 Commitments

A *commitment scheme* is a function that, given a *commitment trapdoor* generated at random and an input, can be used to commit to the input in such a way that:

- no information is revealed about it without the trapdoor ("hiding"),
- given the trapdoor and input, the commitment can be verified to *open* to that input and no other ("binding").

A commitment scheme COMM defines a type of inputs $\mathsf{COMM.Input}$, a type of commitments $\mathsf{COMM.Output}$, a type of commitment trapdoors $\mathsf{COMM.Trapdoor}$, and a trapdoor generator $\mathsf{COMM.GenTrapdoor} : () \xrightarrow{\mathrm{R}} \mathsf{COMM.Trapdoor}$.

Let $\mathsf{COMM} : \mathsf{COMM.Trapdoor} \times \mathsf{COMM.Input} \to \mathsf{COMM.Output}$ be a function satisfying the following security requirements.

Security requirements:

- **Computational hiding:** For all $x, x' : \mathsf{COMM.Input}$, distributions $\{\, \mathsf{COMM}_r(x) \mid r \xleftarrow{\mathrm{R}} \mathsf{COMM.GenTrapdoor}() \,\}$ and $\{\, \mathsf{COMM}_r(x') \mid r \xleftarrow{\mathrm{R}} \mathsf{COMM.GenTrapdoor}() \,\}$ are computationally indistinguishable.
- **Computational binding:** It is infeasible to find $x, x' : \mathsf{COMM.Input}$ and $r, r' : \mathsf{COMM.Trapdoor}$ such that $x \neq x'$ and $\mathsf{COMM}_r(x) = \mathsf{COMM}_{r'}(x')$.

Notes:

- $\mathsf{COMM.GenTrapdoor}$ need not produce the uniform distribution on $\mathsf{COMM.Trapdoor}$. In that case, it is incorrect to choose a trapdoor from the latter distribution.
- If it were only feasible to find $x : \mathsf{COMM.Input}$ and $r, r' : \mathsf{COMM.Trapdoor}$ such that $r \neq r'$ and $\mathsf{COMM}_r(x) = \mathsf{COMM}_{r'}(x)$, this would not contradict the computational binding security requirement.

3.3 Pseudo Random Functions

TODO

3.4 Symmetric Encryption

Let Sym be a symmetric encryption scheme with keyspace $\mathsf{Sym.Key}$, encrypting plaintexts in $\mathsf{Sym.Plaintext}$ to produce ciphertexts in $\mathsf{Sym.Ciphertext}$.

$\mathsf{Sym.Encrypt} : \mathsf{Sym.Key} \times \mathsf{Sym.Plaintext} \to \mathsf{Sym.Ciphertext}$ is the encryption algorithm.

$\mathsf{Sym.Decrypt} : \mathsf{Sym.Key} \times \mathsf{Sym.Ciphertext} \to \mathsf{Sym.Plaintext} \cup \{\perp\}$ is the decryption algorithm, such that for any $\mathsf{K} \in \mathsf{Sym.Key}$ and $\mathsf{P} \in \mathsf{Sym.Plaintext}$, $\mathsf{Sym.Decrypt}_{\mathsf{K}}(\mathsf{Sym.Encrypt}_{\mathsf{K}}(\mathsf{P})) = \mathsf{P}$. $\perp$ is used to represent the decryption of an invalid ciphertext.

Security requirements TODO (see [BN2007]).

3.5 Key Agreement

A *key agreement scheme* is a cryptographic protocol in which two parties agree a shared secret, each using their private key and the other party's public key.

A key agreement scheme KA defines a type of public keys $\mathsf{KA.Public}$, a type of private keys $\mathsf{KA.Private}$, and a type of shared secrets $\mathsf{KA.SharedSecret}$. Optionally, it also defines a type $\mathsf{KA.ValidatedPublic} \subseteq \mathsf{KA.Public}$.

Let $\mathsf{KA.DerivePublic} : \mathsf{KA.Private} \times \mathsf{KA.Public} \to \mathsf{KA.Public}$ be a function that derives the KA public key corresponding to a given KA private key and base point.

Let $\mathsf{KA.Agree} : \mathsf{KA.Private} \times \mathsf{KA.Public} \to \mathsf{KA.SharedSecret}$ be the agreement function.

Optional: Let $\mathsf{KA.Base} : \mathsf{KA.Public}$ be a public base point.

Note: The range of $\mathsf{KA.DerivePublic}$ may be a strict subset of $\mathsf{KA.Public}$.

Security requirements:

- The key agreement and the KDF defined in the next section must together satisfy a suitable adaptive security assumption along the lines of [Bernstein2006, section 3] or [ABR1999, Definition 3].

TODO: be more precise about security properties.

3.6 Key Derivation Functions

A Key Derivation Function (KDF) is defined for a particular key agreement scheme and symmetric encryption scheme. It takes the shared secret produced by the key agreement and an additional "context" argument, and derives a key suitable for the encryption scheme.

TODO: can we just use a PRF for this?

3.7 Signature

A *signature scheme* Sig defines:

- a type of signing keys $\mathsf{Sig.Private}$;
- a type of validating keys $\mathsf{Sig.Public}$;
- a type of messages $\mathsf{Sig.Message}$;
- a type of signatures $\mathsf{Sig.Signature}$;
- a randomized signing key generation algorithm $\mathsf{Sig.GenPrivate} : () \xrightarrow{\mathrm{R}} \mathsf{Sig.Private}$;
- an injective validating key derivation algorithm $\mathsf{Sig.DerivePublic} : \mathsf{Sig.Private} \to \mathsf{Sig.Public}$;
- a randomized signing algorithm $\mathsf{Sig.Sign} : \mathsf{Sig.Private} \times \mathsf{Sig.Message} \xrightarrow{\mathrm{R}} \mathsf{Sig.Signature}$;
- a validating algorithm $\mathsf{Sig.Validate} : \mathsf{Sig.Public} \times \mathsf{Sig.Message} \times \mathsf{Sig.Signature} \to \mathbb{B}$;

such that for any signing key $\mathsf{sk} \xleftarrow{\mathrm{R}} \mathsf{Sig.GenPrivate}()$ and corresponding validating key $\mathsf{vk} = \mathsf{Sig.DerivePublic}(\mathsf{sk})$, and any $m : \mathsf{Sig.Message}$ and $s : \mathsf{Sig.Signature} \xleftarrow{\mathrm{R}} \mathsf{Sig.Sign}_{\mathsf{sk}}(m)$, $\mathsf{Sig.Validate}_{\mathsf{vk}}(m, s) = 1$.

TODO: security properties, e.g. existential and strong unforgeability under chosen message attack.

3.7.1 Signature with Re-Randomizable Keys

A "signature scheme with re-randomizable keys" Sig is a signature scheme that additionally defines:

- a type of randomizers $\mathsf{Sig.Random}$;
- a randomizer generator $\mathsf{Sig.GenRandom} : () \xrightarrow{\mathrm{R}} \mathsf{Sig.Random}$;
- a private key randomization algorithm $\mathsf{Sig.RandomizePrivate} : \mathsf{Sig.Random} \times \mathsf{Sig.Private} \to \mathsf{Sig.Private}$;
- a public key randomization algorithm $\mathsf{Sig.RandomizePublic} : \mathsf{Sig.Random} \times \mathsf{Sig.Public} \to \mathsf{Sig.Public}$;
- a distinguished "identity" randomizer $\mathcal{O}_{\mathsf{Sig.Random}} : \mathsf{Sig.Random}$

such that:

- for any $\alpha : \mathsf{Sig.Random}$, $\mathsf{Sig.RandomizePrivate}_\alpha : \mathsf{Sig.Private} \to \mathsf{Sig.Private}$ is injective and easily invertible;
- $\mathsf{Sig.RandomizePrivate}_{\mathcal{O}_{\mathsf{Sig.Random}}}$ is the identity function on $\mathsf{Sig.Private}$.
- for any $\mathsf{sk} : \mathsf{Sig.Private}$,
  $\mathsf{Sig.RandomizePrivate}(\alpha, \mathsf{sk}) : \alpha \xleftarrow{\mathrm{R}} \mathsf{Sig.GenRandom}()$
  is identically distributed to $\mathsf{Sig.GenPrivate}()$.
- for any $\mathsf{sk} : \mathsf{Sig.Private}$ and $\alpha : \mathsf{Sig.Random}$,
  $\mathsf{Sig.RandomizePublic}(\alpha, \mathsf{Sig.DerivePublic}(\mathsf{sk})) = \mathsf{Sig.DerivePublic}(\mathsf{Sig.RandomizePrivate}(\alpha, \mathsf{sk}))$.

The following security requirement for such signature schemes is based on that given in [FKMSSS2016, section 3]. Note that we require Strong Unforgeability with Re-randomized Keys, not Existential Unforgeability with Re-randomized Keys (the latter is called "Unforgeability under Re-randomized Keys" in [FKMSSS2016, Definition 8]).

Security requirement: Strong Unforgeability with Re-randomized Keys under adaptive Chosen Message Attack (SURK-CMA)

For any $\mathsf{sk} : \mathsf{Sig.Private}$, let

$\mathsf{O}_{\mathsf{sk}} : \mathsf{Sig.Message} \times \mathsf{Sig.Random} \to \mathsf{Sig.Signature}$

be a signing oracle with state $Q : \mathscr{P}\big(\mathsf{Sig.Message} \times \mathsf{Sig.Signature}\big)$ initialized to $\{\}$ that records queried messages and corresponding signatures.

$\mathsf{O}_{\mathsf{sk}} := \text{var } Q \leftarrow \{\} \text{ in } (m : \mathsf{Sig.Message}, \alpha : \mathsf{Sig.Random}) \mapsto$
  let $\sigma = \mathsf{Sig.Sign}_{\mathsf{Sig.RandomizePrivate}(\alpha,\mathsf{sk})}(m)$
  $Q \leftarrow Q \cup \{(m, \sigma)\}$
  return $\sigma : \mathsf{Sig.Signature}$.

For random $\mathsf{sk} \xleftarrow{\text{R}} \mathsf{Sig.GenPrivate}()$ and $\mathsf{vk} = \mathsf{Sig.DerivePublic}(\mathsf{sk})$, it must be infeasible for an adversary given vk and a new instance of O_{sk} to find (m', σ', α') such that $\mathsf{Sig.Validate}_{\mathsf{Sig.RandomizePublic}(\alpha',\mathsf{vk})}(m', \sigma') = 1$ and $(m', \sigma') \notin \mathsf{O}_{\mathsf{sk}}.Q$.

Non-normative notes:

- The randomizer and key arguments to Sig.RandomizePrivate and Sig.RandomizePublic are swapped relative to [FKMSSS2016, section 3].
- The requirement for the identity randomizer $\mathcal{O}_{\mathsf{Sig.Random}}$ simplifies the definition of SURK-CMA by removing the need for two oracles (because the oracle for original keys, called O_1 in [FKMSSS2016], is a special case of the oracle for randomized keys).
- Since $\mathsf{Sig.RandomizePrivate}(\alpha, \mathsf{sk}) : \alpha \xleftarrow{\text{R}} \mathsf{Sig.Random}$ has an identical distribution to Sig.GenPrivate(), and since Sig.DerivePublic is a deterministic function, the combination of a re-randomized public key and signature(s) under that key do not reveal the key from which it was re-randomized.
- Since $\mathsf{Sig.RandomizePrivate}_\alpha$ is injective and easily invertible, knowledge of $\mathsf{Sig.RandomizePrivate}(\alpha, \mathsf{sk})$ *and* α implies knowledge of sk.

3.7.2 Signature with Signing Key to Validating Key Monomorphism

A *signature scheme with signing key to validating key monomorphism* Sig is a signature scheme that additionally defines:

- an abelian group on signing keys, with operation $\boxplus$: $\mathsf{Sig.Private} \times \mathsf{Sig.Private} \to \mathsf{Sig.Private}$ and identity $\mathcal{O}_{\boxplus}$;
- an abelian group on validating keys, with operation $\diamondplus$: $\mathsf{Sig.Public} \times \mathsf{Sig.Public} \to \mathsf{Sig.Public}$ and identity $\mathcal{O}_{\diamondplus}$.

such that for any $\mathsf{sk}_{1..2} : \mathsf{Sig.Private}$, $\mathsf{Sig.DerivePublic}(\mathsf{sk}_1 \boxplus \mathsf{sk}_2) = \mathsf{Sig.DerivePublic}(\mathsf{sk}_1) \diamondplus \mathsf{Sig.DerivePublic}(\mathsf{sk}_2)$.

In other words, Sig.DerivePublic is a monomorphism (that is, an injective homomorphism) from the signing key group to the validating key group.

For $n : \mathbb{N}^+$,

- $\boxplus_{i=1}^{n} \mathsf{sk}_i$ means $\mathsf{sk}_1 \boxplus \mathsf{sk}_2 \boxplus \cdots \boxplus \mathsf{sk}_n$;
- $\diamondplus_{i=1}^{n} \mathsf{vk}_i$ means $\mathsf{vk}_1 \diamondplus \mathsf{vk}_2 \diamondplus \cdots \diamondplus \mathsf{vk}_n$.

When $n = 0$ these yield the appropriate group identity, i.e. $\boxplus_{i=1}^{0} \mathsf{sk}_i = \mathcal{O}_{\boxplus}$ and $\diamondplus_{i=1}^{0} \mathsf{vk}_i = \mathcal{O}_{\diamondplus}$.

$\boxminus\mathsf{sk}$ means the private key such that $(\boxminus\mathsf{sk}) \boxplus \mathsf{sk} = \mathcal{O}_{\boxplus}$, and $\mathsf{sk}_1 \boxminus \mathsf{sk}_2$ means $\mathsf{sk}_1 \boxplus (\boxminus\mathsf{sk}_2)$.

$\diamondminus\mathsf{vk}$ means the public key such that $(\diamondminus\mathsf{vk}) \diamondplus \mathsf{vk} = \mathcal{O}_{\diamondplus}$, and $\mathsf{vk}_1 \diamondminus \mathsf{vk}_2$ means $\mathsf{vk}_1 \diamondplus (\diamondminus\mathsf{vk}_2)$.

With a change of notation from μ to Sig.DerivePublic, $+$ to $\boxplus$, and $\cdot$ to $\diamondplus$, this is similar to the definition of a "Signature with Secret Key to Public Key Homomorphism" in [DS2016, Definition 13], except for an additional requirement for the homomorphism to be injective.

Security requirement: For any $\mathsf{sk}_1 : \mathsf{Sig.Private}$, and an unknown $\mathsf{sk}_2 \xleftarrow{\text{R}} \mathsf{Sig.GenPrivate}()$ chosen independently of sk_1, the distribution of $\mathsf{sk}_1 \boxplus \mathsf{sk}_2$ is computationally indistinguishable from that of Sig.GenPrivate(). (Since $\boxplus$ is an abelian group operation, this implies that for $n : \mathbb{N}^+$, $\boxplus_{i=1}^{n} \mathsf{sk}_i$ is computationally indistinguishable from Sig.GenPrivate() when at least one of $\mathsf{sk}_{1..n}$ is unknown.)

3.8 Represented Group

A *represented group* $\mathbb{G}$ consists of:

- a subgroup order parameter $r_{\mathbb{G}} : \mathbb{N}^+$, which must be prime;
- a cofactor parameter $h_{\mathbb{G}} : \mathbb{N}^+$;
- a group $\mathbb{G}$ of order $h_{\mathbb{G}} \cdot r_{\mathbb{G}}$, written additively with operation $+ : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$, and additive identity $\mathcal{O}_{\mathbb{G}}$;
- a bit-length parameter $\ell_{\mathbb{G}} : \mathbb{N}$;
- a representation function $\mathsf{repr}_{\mathbb{G}} : \mathbb{G} \to \mathbb{B}^{[\ell_{\mathbb{G}}]}$ and an abstraction function $\mathsf{abst}_{\mathbb{G}} : \mathbb{B}^{[\ell_{\mathbb{G}}]} \to \mathbb{G} \cup \{\perp\}$, such that $\mathsf{abst}_{\mathbb{G}}$ is the left inverse of $\mathsf{repr}_{\mathbb{G}}$, i.e. for all $P \in \mathbb{G}$, $\mathsf{abst}_{\mathbb{G}}(\mathsf{repr}_{\mathbb{G}}(P)) = P$, and for all S not in the image of $\mathsf{repr}_{\mathbb{G}}$, $\mathsf{abst}_{\mathbb{G}}(S) = \perp$.

Define $\mathbb{G}^{(r)}$ as the order-$r_{\mathbb{G}}$ subgroup of $\mathbb{G}$, which is called a represented subgroup. Note that this includes $\mathcal{O}_{\mathbb{G}}$. For the set of points of order $r_{\mathbb{G}}$ (which excludes $\mathcal{O}_{\mathbb{G}}$), we write $\mathbb{G}^{(r)*}$.

Define $\mathbb{G}^{(r)}_{\star} := \{\mathsf{repr}_{\mathbb{G}}(P) : \mathbb{B}^{[\ell_{\mathbb{G}}]} \mid P \in \mathbb{G}^{(r)}\}$.

For $G : \mathbb{G}$ we write $-G$ for the negation of G, such that $(-G) + G = \mathcal{O}_{\mathbb{G}}$. We write $G - H$ for $G + (-H)$.

We also extend the $\sum$ notation to addition on group elements.

For $G : \mathbb{G}$ and $k : \mathbb{Z}$ we write $[k]\,G$ for scalar multiplication on the group, i.e.

$$[k]\,G := \begin{cases} \sum_{i=1}^{k} G, & \text{if } k \geq 0 \\ \sum_{i=1}^{-k} (-G), & \text{otherwise.} \end{cases}$$

For $G : \mathbb{G}$ and $a : \mathbb{F}_{r_{\mathbb{G}}}$, we may also write $[a]\,G$ meaning $[a \bmod r_{\mathbb{G}}]\,G$ as defined above. (This variant is not defined for fields other than $\mathbb{F}_{r_{\mathbb{G}}}$.)

3.8.1 Hash Extractor

A *hash extractor* for a represented group $\mathbb{G}$ is a function $\mathsf{Extract}_{\mathbb{G}^{(r)}} : \mathbb{G}^{(r)} \to T$ for some type T, such that $\mathsf{Extract}_{\mathbb{G}^{(r)}}$ is injective on $\mathbb{G}^{(r)}$ (the subgroup of $\mathbb{G}$ of order $r_{\mathbb{G}}$).

Note: Unlike the representation function $\mathsf{repr}_{\mathbb{G}}$, $\mathsf{Extract}_{\mathbb{G}^{(r)}}$ need not have an efficiently computable left inverse.

3.8.2 Group Hash

Given a represented subgroup $\mathbb{G}^{(r)}$, a *family of group hashes into the subgroup*, denoted $\mathsf{GroupHash}^{\mathbb{G}^{(r)}}$, consists of:

- a type $\mathsf{GroupHash.URSType}$ of Uniform Random Strings;
- a type $\mathsf{GroupHash.Input}$ of inputs;
- a function $\mathsf{GroupHash}^{\mathbb{G}^{(r)}} : \mathsf{GroupHash.URSType} \times \mathsf{GroupHash.Input} \to \mathbb{G}^{(r)}$.

Security requirement: For a randomly selected $\mathsf{URS} : \mathsf{GroupHash.URSType}$, it must be reasonble to model $\mathsf{GroupHash}^{\mathbb{G}^{(r)}}_{\mathsf{URS}}$ (restricted to inputs for which it does not return $\perp$) as a random oracle.

URS should be chosen verifiably at random, *after* fixing the concrete group hash algorithm to be used. This mitigates the possibility that the group hash algorithm could have been backdoored.

3.9 Represented Pairing

A *represented pairing* $\mathbb{P}$ consists of:

- a group order parameter $r_{\mathbb{P}} : \mathbb{N}^+$ which must be prime;
- two represented subgroups $\mathbb{P}_{1,2}^{(r)}$, both of order $r_{\mathbb{P}}$;
- a group $\mathbb{P}_T^{(r)}$ of order $r_{\mathbb{P}}$, written multiplicatively with operation $\cdot : \mathbb{P}_T^{(r)} \times \mathbb{P}_T^{(r)} \to \mathbb{P}_T^{(r)}$ and group identity $\mathbf{1}_{\mathbb{P}}$;
- three generators $\mathcal{P}_{\mathbb{P}_{1,2,T}}$ of $\mathbb{P}_{1,2,T}^{(r)}$ respectively;
- a pairing function $\hat{e}_{\mathbb{P}} : \mathbb{P}_1^{(r)} \times \mathbb{P}_2^{(r)} \to \mathbb{P}_T^{(r)}$ satisfying:
  - (Bilinearity) for all $a, b : \mathbb{F}_r^*$, $P : \mathbb{P}_1^{(r)}$, and $Q : \mathbb{P}_2^{(r)}$, $\hat{e}_{\mathbb{P}}([a]\,P, [b]\,Q) = \hat{e}_{\mathbb{P}}(P, Q)^{a \cdot b}$; and
  - (Nondegeneracy) there does not exist $P : \mathbb{P}_1^{(r)*}$ such that for all $Q : \mathbb{P}_2^{(r)}$, $\hat{e}_{\mathbb{P}}(P, Q) = \mathbf{1}_{\mathbb{P}}$.

TODO: align with IEEE 1363.3-2013.

4 Elliptic Curves

An *elliptic curve* is a kind of *plane algebraic curve* over a field. It defines a group of *points* with coordinates in the field that satisfy a *curve equation*. Possible forms of this equation are called *elliptic curve shapes*.

In cryptography, elliptic curves are commonly used to instantiate cryptographic groups.

In this standard we specify the use of elliptic curves of three shapes:

- complete twisted Edwards curves;
- Montgomery curves;
- short Weierstrass curves.

In each of the definitions below, let $\mathbb{F}_q$ be a field such that $q = p^k$ for prime p. $\mathbb{F}_q$ is "non-binary" if $p > 2$.

A *complete twisted Edwards curve*, as defined in [BL2017, section 4.3.4], is an elliptic curve E over a non-binary $\mathbb{F}_q$, parameterized by distinct $a, d : \mathbb{F}_q \setminus \{0\}$ such that a is square and d is nonsquare, with equation $E : a \cdot u^2 + v^2 = 1 + d \cdot u^2 \cdot v^2$. We use the abbreviation "ctEdwards" to refer to complete twisted Edwards curves and coordinates.

A *Montgomery curve*, as defined in [BL2017, section 4.3], is an elliptic curve E over a non-binary field $\mathbb{F}_q$, parameterized by $A, B : \mathbb{F}_q$ such that $B \cdot (A^2 - 4) \neq 0$, with equation $E : B \cdot y^2 = x^3 + A \cdot x^2 + x$.

A *short Weierstrass curve* over a field $\mathbb{F}_q$ with $p > 3$, is an elliptic curve over $\mathbb{F}_q$ parameterized by $a, b : \mathbb{F}_q$ such that $4 \cdot a^3 + 27 \cdot b^2 \neq 0$, with equation $E : y^2 = x^3 + a \cdot x + b$. TODO: reference

TODO: explain birational equivalence between ctEdwards and Montgomery curves.

The above definitions give the curve equations in terms of *affine coordinates*. Other *coordinate systems* are possible for a given curve shape. These are typically designed to reduce the number of field inversions required to compute operations in the elliptic curve group, since inversions can be expensive to compute. However, within zk-SNARK circuits, it is often the case that field inversions have similar cost to field multiplications, which favours the use of affine coordinates.

TODO: the use of other coordinate systems outside circuits does not typically affect interoperability, but may affect security. Consider giving references/advice for each curve shape.

Following the notation in [BL2017] we use (u, v) for affine coordinates on ctEdwards curves, and (x, y) for affine coordinates on Montgomery or short Weierstrass curves.

A point P is normally represented by two $\mathbb{F}_q$ variables, which we name as (P^u, P^v) for an affine ctEdwards point, for instance.

TODO: define group operations, scalar multiplication, and the elliptic curve discrete logarithm problem.

Implementations of scalar multiplication will require the scalar to be represented as a bit sequence. We therefore allow the notation $[k\star]\,P$ meaning $[\mathsf{LEBS2IP}_{\mathsf{length}(k\star)}(k\star)]\,P$. There will be no ambiguity because variables representing bit sequences are named with a $\star$ suffix.

We define the following types representing affine ctEdwards, affine Montgomery, and affine short Weierstrass coordinates respectively:

$$\begin{aligned}
\mathsf{AffineCtEdwards} &:= (u : \mathbb{F}_q) \times (v : \mathbb{F}_q) : a \cdot u^2 + v^2 = 1 + d \cdot u^2 \cdot v^2 \\
\mathsf{AffineMontgomery} &:= (x : \mathbb{F}_q) \times (y : \mathbb{F}_q) : B \cdot y^2 = x^3 + A \cdot x^2 + x \\
\mathsf{AffineShortWeierstrass} &:= (x : \mathbb{F}_q) \times (y : \mathbb{F}_q) : y^2 = x^3 + a \cdot x + b
\end{aligned}$$

We also define types representing compressed, *not necessarily valid*, ctEdwards, Montgomery, and short Weierstrass coordinates:

$$\begin{aligned}
\mathsf{CompressedCtEdwards} &:= (\tilde{u} : \mathbb{B}) \times (v : \mathbb{F}_q) \\
\mathsf{CompressedMontgomery} &:= (x : \mathbb{F}_q) \times (\tilde{y} : \mathbb{B}) \\
\mathsf{CompressedShortWeierstrass} &:= (x : \mathbb{F}_q) \times (\tilde{y} : \mathbb{B})
\end{aligned}$$

See TODO for how these types are represented as byte sequences in external encodings.

An important consideration when using affine Montgomery or affine short Weierstrass coordinates is that the addition formulae for these coordinate systems are not complete, that is, there are cases for which they do not apply. To avoid obtaining wrong answers, we must either check for these cases or ensure that they do not arise.

We will need the theorem below about y-coordinates of points on certain Montgomery curves.

Theorem 4.0.1. *Let $P = (x, y)$ be a point other than $(0, 0)$ on a Montgomery curve $E_{\mathsf{Montgomery}(A,B)}$ over $\mathbb{F}_r$, such that $A^2 - 4$ is a nonsquare in $\mathbb{F}_r$. Then $y \neq 0$.*

Proof. Substituting $y = 0$ into the Montgomery curve equation gives $0 = x^3 + A \cdot x^2 + x = x \cdot (x^2 + A \cdot x + 1)$. So either $x = 0$ or $x^2 + A \cdot x + 1 = 0$. Since $P \neq (0, 0)$, the case $x = 0$ is excluded. In the other case, complete the square for $x^2 + A \cdot x + 1 = 0$ to give the equivalent $(2 \cdot x + A)^2 = A^2 - 4$. The left-hand side is a square, so if the right-hand side is a nonsquare, then there are no solutions for x. □

4.1 Elliptic curves as Represented Groups

TODO

4.2 Pairing-friendly elliptic curves

TODO

5 Integers, Bit Sequences, and Endianness

The following functions convert between sequences of bits, sequences of bytes, and integers:

- $\mathsf{I2LEBSP} : (\ell : \mathbb{N}) \times \{0 \mathinner{..} 2^\ell - 1\} \to \mathbb{B}^{[\ell]}$, such that $\mathsf{I2LEBSP}_\ell(x)$ is the sequence of ℓ bits representing x in little-endian order;
- $\mathsf{LEBS2IP} : (\ell : \mathbb{N}) \times \mathbb{B}^{[\ell]} \to \{0 \mathinner{..} 2^\ell - 1\}$ such that $\mathsf{LEBS2IP}_\ell(S)$ is the integer represented in little-endian order by the bit sequence S of length ℓ.
- $\mathsf{LEOS2IP} : (\ell : \mathbb{N} \mid \ell \bmod 8 = 0) \times \mathbb{B}^{\mathbb{Y}^{[\ell/8]}} \to \{0 \mathinner{..} 2^\ell - 1\}$ such that $\mathsf{LEOS2IP}_\ell(S)$ is the integer represented in little-endian order by the byte sequence S of length $\ell/8$.

- $\mathsf{LEBS2OSP} : (\ell : \mathbb{N}) \times \mathbb{B}^{[\ell]} \to \mathbb{B}^{\mathbb{Y}[\mathsf{ceiling}(\ell/8)]}$ defined as follows: pad the input on the right with $8 \cdot \mathsf{ceiling}(\ell/8) - \ell$ zero bits so that its length is a multiple of 8 bits. Then convert each group of 8 bits to a byte value with the *least* significant bit first, and concatenate the resulting bytes in the same order as the groups.
- $\mathsf{LEOS2BSP} : (\ell : \mathbb{N} \mid \ell \bmod 8 = 0) \times \mathbb{B}^{\mathbb{Y}[\mathsf{ceiling}(\ell/8)]} \to \mathbb{B}^{[\ell]}$ defined as follows: convert each byte to a group of 8 bits with the *least* significant bit first, and concatenate the resulting groups in the same order as the bytes.

6 Concrete Cryptographic Schemes

6.1 Hash Functions

6.1.1 BLAKE2 Hash Functions

BLAKE2 is defined by [ANWW2013]. It has BLAKE2b and BLAKE2s variants.

$\mathsf{BLAKE2b}\text{-}\ell(p, x)$ refers to unkeyed BLAKE2b in sequential mode, with an output digest length of $\ell/8$ bytes, 16-byte personalization string p, and input x.

$\mathsf{BLAKE2b}\text{-}\ell : \mathbb{B}^{\mathbb{Y}[16]} \times \mathbb{B}^{\mathbb{Y}[\mathbb{N}]} \to \mathbb{B}^{\mathbb{Y}[\ell/8]}$

$\mathsf{BLAKE2s}\text{-}\ell(p, x)$ refers to unkeyed BLAKE2s in sequential mode, with an output digest length of $\ell/8$ bytes, 8-byte personalization string p, and input x.

$\mathsf{BLAKE2s}\text{-}\ell : \mathbb{B}^{\mathbb{Y}[8]} \times \mathbb{B}^{\mathbb{Y}[\mathbb{N}]} \to \mathbb{B}^{\mathbb{Y}[\ell/8]}$

Note: $\mathsf{BLAKE2b}\text{-}\ell$ is not the same as $\mathsf{BLAKE2b\text{-}512}$ truncated to ℓ bits, and similarly $\mathsf{BLAKE2s}\text{-}\ell$ is not the same as $\mathsf{BLAKE2s\text{-}256}$ truncated to ℓ bits, because the digest length is encoded in the parameter block.

6.1.2 Bowe–Hopwood Pedersen Hash Function

$\mathsf{PedersenHash}$ is an algebraic hash function with collision resistance (for fixed input length) derived from assumed hardness of the Discrete Logarithm Problem on an elliptic curve. It is based on the work of David Chaum, Ivan Damgård, Jeroen van de Graaf, Jurjen Bos, George Purdy, Eugène van Heijst and Birgit Pfitzmann in [CDvdG1987], [BCP1988] and [CvHP1991], and of Mihir Bellare, Oded Goldreich, and Shafi Goldwasser in [BGG1995], with optimizations for efficient instantiation in zk-SNARK circuits by Sean Bowe and Daira Hopwood.

Let $\mathbb{G}$ be a ctEdwards curve over a field $\mathbb{F}_q$, and let $\mathbb{G}^{(r)}$, $\mathcal{O}_{\mathbb{G}}$, $q_{\mathbb{G}}$, $r_{\mathbb{G}}$, $a_{\mathbb{G}}$, and $d_{\mathbb{G}}$ be as defined in Section 4.1.

Let $\mathsf{Extract}_{\mathbb{G}^{(r)}} : \mathbb{G}^{(r)} \to \mathbb{F}_q$ be as defined in Section 6.5.4.

Let $\mathsf{FindGroupHash}^{\mathbb{G}^{(r)*}}$ be as defined in Section 6.5.5.

Let c be the largest integer such that $4 \cdot \frac{2^{4 \cdot c}}{15} \leq \frac{r_{\mathbb{G}} - 1}{2}$.

Define $\mathcal{I} : \mathbb{B}^{\mathbb{Y}[8]} \times \mathbb{N} \to \mathbb{G}^{(r)*}$ by:

$\mathcal{I}_i^D := \mathsf{FindGroupHash}^{\mathbb{G}^{(r)*}}(D, \mathsf{LEBS2OSP}_{32}(\mathsf{I2LEBSP}_{32}(i-1)))$.

Define $\mathsf{PedersenHashToPoint}(D : \mathbb{B}^{\mathbb{Y}[8]}, M : \mathbb{B}^{[\mathbb{N}^+]}) \to \mathbb{G}^{(r)}$ as follows:

Pad M to a multiple of 3 bits by appending zero bits, giving M'.

Let $n = \mathsf{ceiling}\left(\frac{\mathsf{length}(M')}{3 \cdot c}\right)$.

Split M' into n *segments* $M_{1\,..\,n}$ so that $M' = \mathsf{concat}_{\mathbb{B}}(M_{1\,..\,n})$, and each of $M_{1\,..\,n-1}$ is of length $3 \cdot c$ bits. (M_n may be shorter.)

Return $\sum_{i=1}^{n} [\langle M_i \rangle] \mathcal{I}_i^D : \mathbb{G}^{(r)}$.

where $\langle \bullet \rangle : \mathbb{B}^{[3 \cdot \{1\,..\,c\}]} \to \{-\frac{r_{\mathbb{G}}-1}{2}\,..\,\frac{r_{\mathbb{G}}-1}{2}\} \setminus \{0\}$ is defined as:

Let $k_i = \mathsf{length}(M_i)/3$.

Split M_i into 3-bit *chunks* $m_{1\,..\,k_i}$ so that $M_i = \mathsf{concat}_{\mathbb{B}}(m_{1\,..\,k_i})$.

Write each m_j as $[s_0^j, s_1^j, s_2^j]$, and let $\mathsf{enc}(m_j) = (1 - 2 \cdot s_2^j) \cdot (1 + s_0^j + 2 \cdot s_1^j) : \mathbb{Z}$.

Let $\langle M_i \rangle = \sum_{j=1}^{k_i} \mathsf{enc}(m_j) \cdot 2^{4 \cdot (j-1)}$.

Finally, define $\mathsf{PedersenHash} : \mathbb{B}^{\mathbb{Y}^{[8]}} \times \mathbb{B}^{[\mathbb{N}^+]} \to \mathbb{B}^{[\ell_{\mathbb{G}}]}$ by:

$\mathsf{PedersenHash}(D, M) := \mathsf{Extract}_{\mathbb{G}^{(r)}}\big(\mathsf{PedersenHashToPoint}(D, M)\big)$.

See Section B.3.9 for rationale and efficient circuit implementation of these functions.

Assuming hardness of the Discrete Logarithm Problem on $\mathbb{G}$, PedersenHash and PedersenHashToPoint are collision resistant between inputs *of fixed length*, for a given fixed personalization input D.

TODO: collision resistance should hold across differing D given a suitable assumption on $\mathsf{FindGroupHash}^{\mathbb{G}^{(r)*}}$.

Non-normative note: These hash functions are *not* collision resistant for variable-length inputs, and no other security properties commonly associated with hash functions are guaranteed.

Lemma 6.1.1. *If* $\mathsf{Extract}_{\mathbb{G}^{(r)}}$ *is injective, then the encoding function* $\langle \bullet \rangle$ *is injective.*

Proof. We first check that the range of $\sum_{j=1}^{k_i} \mathsf{enc}(m_j) \cdot 2^{4 \cdot (j-1)}$ is a subset of the allowable range $\{-\frac{r_{\mathbb{G}}-1}{2}\,..\,\frac{r_{\mathbb{G}}-1}{2}\} \setminus \{0\}$. The range of this expression is a subset of $\{-\Delta\,..\,\Delta\} \setminus \{0\}$ where $\Delta = 4 \cdot \sum_{i=1}^{c} 2^{4 \cdot (i-1)} = 4 \cdot \frac{2^{4 \cdot c}-1}{15}$. This is met by definition for the specified c. This implies that there is no "wrap around" and so $\sum_{j=1}^{k_i} \mathsf{enc}(m_j) \cdot 2^{4 \cdot (j-1)}$ may be treated as an integer expression.

enc is injective. In order to prove that $\langle \bullet \rangle$ is injective, consider $\langle \bullet \rangle^{\Delta} : \mathbb{B}^{[3 \cdot \{1\,..\,c\}]} \to \{0\,..\,2 \cdot \Delta\}$ such that $\langle M_i \rangle^{\Delta} = \langle M_i \rangle + \Delta$. With k_i and m_j defined as above, we have $\langle M_i \rangle^{\Delta} = \sum_{j=1}^{k_i} \mathsf{enc}'(m_j) \cdot 2^{4 \cdot (j-1)}$ where $\mathsf{enc}'(m_j) = \mathsf{enc}(m_j) + 4$ is in $\{0\,..\,8\}$ and enc' is injective. Express this sum in hexadecimal; then each m_j affects only one hex digit, and it is easy to see that $\langle \bullet \rangle^{\Delta}$ is injective. Therefore so is $\langle \bullet \rangle$. □

Since the security proof from [BGG1995, Appendix A] depends only on the encoding being injective and its range not including zero, the proof can be adapted straightforwardly to show that PedersenHashToPoint is collision resistant under the same assumptions and security bounds. Since $\mathsf{Extract}_{\mathbb{G}^{(r)}}$ is assumed to be injective, it follows that PedersenHash is equally collision resistant.

6.2 BLAKE2

TODO: use of BLAKE2 as a general hash function, PRF, and commitment scheme.

6.3 RedDSA

RedDSA is a Schnorr-based signature scheme, optionally supporting key re-randomization as described in Section 3.7.1. It also supports a signing key to validating key monomorphism as described in Section 3.7.2. It is based on a scheme from [FKMSSS2016, section 3], with some ideas from EdDSA [BJLSY2015].

An implementation of RedDSA may be specialized to an elliptic curve or to other parameters. For example, RedJubjub is a specialization of RedDSA to the Jubjub curve (Section 6.5.3), using the BLAKE2b-512 hash function.

We first describe the scheme RedDSA over a general represented group. Its parameters are:

- a represented group $\mathbb{G}$, which also defines a subgroup $\mathbb{G}^{(r)}$ of order $r_{\mathbb{G}}$, a cofactor $h_{\mathbb{G}}$, a group operation $+$, an additive identity $\mathcal{O}_{\mathbb{G}}$, a bit-length $\ell_{\mathbb{G}}$, a representation function $\mathsf{repr}_{\mathbb{G}}$, and an abstraction function $\mathsf{abst}_{\mathbb{G}}$, as specified in Section 3.8;
- $\mathcal{P}_{\mathbb{G}}$, a generator of $\mathbb{G}^{(r)}$;
- a bit-length $\ell_{\mathsf{H}} : \mathbb{N}$ such that $2^{\ell_{\mathsf{H}}-128} \geq r_{\mathbb{G}}$ and $\ell_{\mathsf{H}} \bmod 8 = 0$;
- a cryptographic hash function $\mathsf{H} : \mathbb{B}^{\mathbb{Y}[\mathbb{N}]} \to \mathbb{B}^{\mathbb{Y}[\ell_{\mathsf{H}}/8]}$.

Its associated types are defined as follows:

$\mathsf{RedDSA.Message} := \mathbb{B}^{\mathbb{Y}[\mathbb{N}]}$

$\mathsf{RedDSA.Signature} := \mathbb{B}^{\mathbb{Y}[\mathsf{ceiling}(\ell_{\mathbb{G}}/8) + \mathsf{ceiling}(\mathsf{bitlength}(r_{\mathbb{G}})/8)]}$

$\mathsf{RedDSA.Public} := \mathbb{G}$

$\mathsf{RedDSA.Private} := \mathbb{F}_{r_{\mathbb{G}}}$.

$\mathsf{RedDSA.Random} := \mathbb{F}_{r_{\mathbb{G}}}$.

Define $\mathsf{H}^{\circledast} : \mathbb{B}^{\mathbb{Y}[\mathbb{N}]} \to \mathbb{F}_{r_{\mathbb{G}}}$ by:

$\mathsf{H}^{\circledast}(B) = \mathsf{LEOS2IP}_{\ell_{\mathsf{H}}}\big(\mathsf{H}(B)\big) \pmod{r_{\mathbb{G}}}$

Define $\mathsf{RedDSA.GenPrivate} : () \xrightarrow{\mathrm{R}} \mathsf{RedDSA.Private}$ as:

Return $\mathsf{sk} \xleftarrow{\mathrm{R}} \mathbb{F}_{r_{\mathbb{G}}}$.

Define $\mathsf{RedDSA.DerivePublic} : \mathsf{RedDSA.Private} \to \mathsf{RedDSA.Public}$ by:

$\mathsf{RedDSA.DerivePublic}(\mathsf{sk}) := [\mathsf{sk}]\,\mathcal{P}_{\mathbb{G}}$.

Define $\mathsf{RedDSA.GenRandom} : () \xrightarrow{\mathrm{R}} \mathsf{RedDSA.Random}$ as:

Choose a byte sequence T uniformly at random on $\mathbb{B}^{\mathbb{Y}[(\ell_{\mathsf{H}}+128)/8]}$.

Return $\mathsf{H}^{\circledast}(T)$.

Define $\mathcal{O}_{\mathsf{RedDSA.Random}} := 0 \pmod{r_{\mathbb{G}}}$.

Define $\mathsf{RedDSA.RandomizePrivate} : \mathsf{RedDSA.Random} \times \mathsf{RedDSA.Private} \to \mathsf{RedDSA.Private}$ by:

$\mathsf{RedDSA.RandomizePrivate}(\alpha, \mathsf{sk}) := \mathsf{sk} + \alpha \pmod{r_{\mathbb{G}}}$.

Define $\mathsf{RedDSA.RandomizePublic} : \mathsf{RedDSA.Random} \times \mathsf{RedDSA.Public} \to \mathsf{RedDSA.Public}$ as:

$\mathsf{RedDSA.RandomizePublic}(\alpha, \mathsf{vk}) := \mathsf{vk} + [\alpha]\,\mathcal{P}_{\mathbb{G}}$.

Define $\mathsf{RedDSA.Sign} : (\mathsf{sk} : \mathsf{RedDSA.Private}) \times (M : \mathsf{RedDSA.Message}) \xrightarrow{\mathrm{R}} \mathsf{RedDSA.Signature}$ as:

Choose a byte sequence T uniformly at random on $\mathbb{B}^{\mathbb{Y}[(\ell_{\mathsf{H}}+128)/8]}$.

Let $\underline{\mathsf{vk}} = \mathsf{LEBS2OSP}_{\ell_{\mathbb{G}}}(\mathsf{repr}_{\mathbb{G}}(\mathsf{RedDSA.DerivePublic}(\mathsf{sk})))$.

Let $r = \mathsf{H}^{\circledast}(T \,||\, \underline{\mathsf{vk}} \,||\, M)$.

Let $R = [r]\,\mathcal{P}_{\mathbb{G}}$.

Let $\underline{R} = \mathsf{LEBS2OSP}_{\ell_{\mathbb{G}}}(\mathsf{repr}_{\mathbb{G}}(R))$.

Let $S = (r + \mathsf{H}^{\circledast}(\underline{R} \,||\, \underline{\mathsf{vk}} \,||\, M) \cdot \mathsf{sk}) \bmod r_{\mathbb{G}}$.

Let $\underline{S} = \mathsf{LEBS2OSP}_{\mathsf{bitlength}(r_{\mathbb{G}})}(\mathsf{I2LEBSP}_{\mathsf{bitlength}(r_{\mathbb{G}})}(S))$.

Return $\underline{R} \,||\, \underline{S}$.

Define $\mathsf{RedDSA.Validate} : (\mathsf{vk} : \mathsf{RedDSA.Public}) \times (M : \mathsf{RedDSA.Message}) \times (\sigma : \mathsf{RedDSA.Signature}) \to \mathbb{B}$ as:

Let $\underline{R}$ be the first $\mathsf{ceiling}(\ell_{\mathbb{G}}/8)$ bytes of σ, and let $\underline{S}$ be the remaining $\mathsf{ceiling}(\mathsf{bitlength}(r_{\mathbb{G}})/8)$ bytes.

Let $R = \mathsf{abst}_{\mathbb{G}}\big(\mathsf{LEOS2BSP}_{\ell_{\mathbb{G}}}(\underline{R})\big)$, and let $S = \mathsf{LEOS2IP}_{8 \cdot \mathsf{length}(\underline{S})}(\underline{S})$.

Let $\underline{\mathsf{vk}} = \mathsf{LEBS2OSP}_{\ell_{\mathbb{G}}}(\mathsf{repr}_{\mathbb{G}}(\mathsf{vk}))$.

Let $c = \mathsf{H}^{\circledast}(\underline{R} \,||\, \underline{\mathsf{vk}} \,||\, M)$.

Return 1 if $R \neq \perp$ and $S < r_{\mathbb{G}}$ and $[h_{\mathbb{G}}]\left(-[S]\,\mathcal{P}_{\mathbb{G}} + R + [c]\,\mathsf{vk}\right) = \mathcal{O}_{\mathbb{G}}$, otherwise 0.

Notes:

- The verification algorithm *does not* check that R is a point of order at least $r_{\mathbb{G}}$. It *does* check that $\underline{R}$ is the canonical representation (as output by $\mathsf{repr}_{\mathbb{G}}$) of a point on the curve.
- Appendix Section C.1 describes an optimization that can be used to speed up verification of batches of RedDSA signatures.

Non-normative note: The randomization used in RedDSA.RandomizePrivate and RedDSA.RandomizePublic may interact with other uses of additive properties of keys for Schnorr-based signature schemes, and so careful analysis of potential interactions is required if these properties are used.

The two abelian groups specified in Section 3.7.2 are instantiated for RedDSA as follows:

- $\mathcal{O}_{\boxplus} := 0 \pmod{r_{\mathbb{G}}}$
- $\mathsf{sk}_1 \boxplus \mathsf{sk}_2 := \mathsf{sk}_1 + \mathsf{sk}_2 \pmod{r_{\mathbb{G}}}$
- $\mathcal{O}_{\oplus} := \mathcal{O}_{\mathbb{G}}$
- $\mathsf{vk}_1 \oplus \mathsf{vk}_2 := \mathsf{vk}_1 + \mathsf{vk}_2$.

As required, RedDSA.DerivePublic is a group monomorphism, since it is injective and:

$$\begin{aligned}\mathsf{RedDSA.DerivePublic}(\mathsf{sk}_1 \boxplus \mathsf{sk}_2) &= [\mathsf{sk}_1 + \mathsf{sk}_2 \pmod{r_{\mathbb{G}}}]\,\mathcal{P}_{\mathbb{G}} \\ &= [\mathsf{sk}_1]\,\mathcal{P}_{\mathbb{G}} + [\mathsf{sk}_2]\,\mathcal{P}_{\mathbb{G}} \quad (\text{since } \mathcal{P}_{\mathbb{G}} \text{ has order } r_{\mathbb{G}}) \\ &= \mathsf{RedDSA.DerivePublic}(\mathsf{sk}_1) \oplus \mathsf{RedDSA.DerivePublic}(\mathsf{sk}_2).\end{aligned}$$

A RedDSA public key vk can be encoded as a bit sequence $\mathsf{repr}_{\mathbb{G}}(\mathsf{vk})$ of length $\ell_{\mathbb{G}}$ bits (or as a corresponding byte sequence $\underline{\mathsf{vk}}$ by then applying $\mathsf{LEBS2OSP}_{\ell_{\mathbb{G}}}$).

The scheme RedJubjub specializes RedDSA with:

- $\mathbb{G} := \mathbb{J}$ as defined in Section 6.5.3;
- $\ell_{\mathsf{H}} := 512$;
- $\mathsf{H}(x) := \mathsf{BLAKE2b\text{-}512}(\texttt{“Zcash\_RedJubjubH”}, x)$ as defined in Section 6.1.1.

The generator $\mathcal{P}_{\mathbb{G}} : \mathbb{G}^{(r)}$ is left as an unspecified parameter.

6.4 Commitment schemes

6.4.1 Windowed Bowe–Hopwood Pedersen commitments

Section 6.1.2 defines the Bowe–Hopwood Pedersen hash construction. We construct *windowed Bowe–Hopwood Pedersen commitments* by reusing that construction, and adding a randomized point on the elliptic curve.

Let $D : \mathbb{B}^{\mathbb{Y}^{[8]}}$ be a personalization parameter.

$$\mathsf{WindowedPedersenCommit}_r(D, s) := \mathsf{PedersenHashToPoint}(D, s) + [r]\,\mathsf{FindGroupHash}^{\mathbb{G}^{(r)*}}(D, \texttt{“r”})$$

6.4.2 Homomorphic Pedersen commitments

The windowed Pedersen commitments defined in the preceding section are highly efficient, but they do not support the homomorphic property needed for some applications.

In order to support this property, we also define *homomorphic Pedersen commitments* as follows:

$\mathsf{HomomorphicPedersenCommit}_{\mathsf{r}}(D, s) := [s]\,\mathsf{FindGroupHash}^{\mathbb{G}^{(r)*}}(D, \text{“}\mathtt{v}\text{”}) + [\mathsf{r}]\,\mathsf{FindGroupHash}^{\mathbb{G}^{(r)*}}(D, \text{“}\mathtt{r}\text{”})$

$\mathsf{ValueCommit.GenTrapdoor}()$ generates the uniform distribution on $\mathbb{F}_{r_{\mathbb{G}}}$.

6.5 Represented Groups and Pairings

6.5.1 BN-254 Represented Pairing

The represented pairing BN-254 is defined in this section.

Let $q_{\mathbb{G}} := 21888242871839275222246405745257275088696311157297823662689037894645226208583$.

Let $r_{\mathbb{G}} := 21888242871839275222246405745257275088548364400416034343698204186575808495617$.

Let $b_{\mathbb{G}} := 3$.

($q_{\mathbb{G}}$ and $r_{\mathbb{G}}$ are prime.)

Let $\mathbb{G}_1^{(r)}$ be the group (of order $r_{\mathbb{G}}$) of rational points on a Barreto–Naehrig ([BN2005]) curve $E_{\mathbb{G}_1}$ over $\mathbb{F}_{q_{\mathbb{G}}}$ with equation $y^2 = x^3 + b_{\mathbb{G}}$. This curve has embedding degree 12 with respect to $r_{\mathbb{G}}$.

Let $\mathbb{G}_2^{(r)}$ be the subgroup of order $r_{\mathbb{G}}$ in the sextic twist $E_{\mathbb{G}_2}$ of $E_{\mathbb{G}_1}$ over $\mathbb{F}_{q_{\mathbb{G}^2}}$ with equation $y^2 = x^3 + \frac{b_{\mathbb{G}}}{\xi}$, where $\xi : \mathbb{F}_{q_{\mathbb{G}^2}}$.

We represent elements of $\mathbb{F}_{q_{\mathbb{G}^2}}$ as polynomials $a_1 \cdot t + a_0 : \mathbb{F}_{q_{\mathbb{G}}}[t]$, modulo the irreducible polynomial $t^2 + 1$; in this representation, ξ is given by $t + 9$.

Let $\mathbb{G}_T^{(r)}$ be the subgroup of $r_{\mathbb{G}}{}^{\text{th}}$ roots of unity in $\mathbb{F}^*_{q_{\mathbb{G}^{12}}}$, with multiplicative identity $\mathbf{1}_{\mathbb{G}}$.

Let $\hat{e}_{\mathbb{G}}$ be the optimal ate pairing (see [Vercauter2009] and [AKLGL2010, section 2]) of type $\mathbb{G}_1^{(r)} \times \mathbb{G}_2^{(r)} \to \mathbb{G}_T^{(r)}$.

For $i : \{1 \,..\, 2\}$, let $\mathcal{O}_{\mathbb{G}_i}$ be the point at infinity (which is the additive identity) in $\mathbb{G}_i^{(r)}$, and let $\mathbb{G}_i^{(r)*} := \mathbb{G}_i^{(r)} \setminus \{\mathcal{O}_{\mathbb{G}_i}\}$.

Let $\mathcal{P}_{\mathbb{G}_1} : \mathbb{G}_1^{(r)*} := (1, 2)$.

Let $\mathcal{P}_{\mathbb{G}_2} : \mathbb{G}_2^{(r)*} := (11559732032986387107991004021392285783925812861821192530917403151452391805634 \cdot t +$
$10857046999023057135944570762232829481370756359578518086990519993285655852781,$
$40823678758634336813322034031454355683168513275934012081057410762141200935 31 \cdot t +$
$84956539231234314176049732474892724384181905872636001487702806493069581019 30)$.

$\mathcal{P}_{\mathbb{G}_1}$ and $\mathcal{P}_{\mathbb{G}_2}$ are generators of $\mathbb{G}_1^{(r)}$ and $\mathbb{G}_2^{(r)}$ respectively.

Define $\mathsf{I2BEBSP} : (\ell : \mathbb{N}) \times \{0 \,..\, 2^{\ell} - 1\} \to \mathbb{B}^{[\ell]}$ as in Section 5.

For a point $P : \mathbb{G}_1^{(r)*} = (x_P, y_P)$:

- The field elements x_P and $y_P : \mathbb{F}_q$ are represented as integers x and $y : \{0 \,..\, q-1\}$.
- Let $\tilde{y} = y \bmod 2$.
- P is encoded as $[\mathtt{0x02} + \tilde{y}] \,||\, \mathsf{I2BEOSP}_{256}(x)$. TODO check

For a point $P : \mathbb{G}_2^{(r)*} = (x_P, y_P)$:

- Define $\mathsf{FE2IP} : \mathbb{F}_{q_{\mathbb{G}}}[t]/(t^2 + 1) \to \{0 \,..\, q_{\mathbb{G}}{}^2 - 1\}$ such that $\mathsf{FE2IP}(a_{w,1} \cdot t + a_{w,0}) = a_{w,1} \cdot q + a_{w,0}$.

- Let $x = \mathsf{FE2IP}(x_P)$, $y = \mathsf{FE2IP}(y_P)$, and $y' = \mathsf{FE2IP}(-y_P)$.
- Let $\tilde{y} = \begin{cases} 1, & \text{if } y > y' \\ 0, & \text{otherwise.} \end{cases}$
- P is encoded as $[\texttt{0x0A} + \tilde{y}] \,||\, \mathsf{I2BEOSP}_{512}(x)$. TODO check

Non-normative notes:

- Only the $r_{\mathbb{G}}$-order subgroups $\mathbb{G}_{2,T}^{(r)}$ are used, not their containing groups $\mathbb{G}_{2,T}$. Points in $\mathbb{G}_2^{(r)*}$ are *always* checked to be of order $r_{\mathbb{G}}$ when decoding from external representation. (The group of rational points $\mathbb{G}_1$ on $E_{\mathbb{G}_1}/\mathbb{F}_{q_{\mathbb{G}}}$ is of order $r_{\mathbb{G}}$ so no subgroup checks are needed in that case, and elements of $\mathbb{G}_T^{(r)}$ are never represented externally.) The (r) superscripts on $\mathbb{G}_{1,2,T}^{(r)}$ are used for consistency with notation elsewhere in this specification.
- The points at infinity $\mathcal{O}_{\mathbb{G}_{1,2}}$ never occur in proofs and have no defined encodings.
- A rational point $P \neq \mathcal{O}_{\mathbb{G}_2}$ on the curve $E_{\mathbb{G}_2}$ can be verified to be of order $r_{\mathbb{G}}$, and therefore in $\mathbb{G}_2^{(r)*}$, by checking that $r_{\mathbb{G}} \cdot P = \mathcal{O}_{\mathbb{G}_2}$.
- TODO: this is what Zcash does, but it's a bit funky. The use of big-endian order by $\mathsf{I2BEBSP}$ is different from the encoding of most other integers in this protocol. The encodings for $\mathbb{G}_{1,2}^{(r)*}$ are consistent with the definition of $\mathsf{EC2OSP}$ for compressed curve points in [IEEE2004, section 5.5.6.2]. The LSB compressed form (i.e. $\mathsf{EC2OSP\text{-}XL}$) is used for points in $\mathbb{G}_1^{(r)*}$, and the SORT compressed form (i.e. $\mathsf{EC2OSP\text{-}XS}$) for points in $\mathbb{G}_2^{(r)*}$.
- Testing $y > y'$ for the compression of $\mathbb{G}_2^{(r)*}$ points is equivalent to testing whether $(a_{y,1}, a_{y,0}) > (a_{-y,1}, a_{-y,0})$ in lexicographic order.
- Algorithms for decompressing points from the above encodings are given in [IEEE2000, Appendix A.12.8] for $\mathbb{G}_1^{(r)*}$, and [IEEE2004, Appendix A.12.11] for $\mathbb{G}_2^{(r)*}$.

When computing square roots in $\mathbb{F}_{q_{\mathbb{G}}}$ or $\mathbb{F}_{q_{\mathbb{G}}{}^2}$ in order to decompress a point encoding, the implementation must not assume that the square root exists, or that the encoding represents a point on the curve.

6.5.2 BLS12-381 Represented Pairing

The represented pairing $\mathsf{BLS12\text{-}381}$ is defined in this section. Parameters are taken from [Bowe2017].

Let $q_{\mathbb{S}} := 4002409555221667393417789825735904156556882819939007885332058136124031650490837864442687629129015664037894272559787$.

Let $r_{\mathbb{S}} := 52435875175126190479447740508185965837690552500527637822603658699938581184513$.

Let $u_{\mathbb{S}} := -15132376222941642752$.

Let $b_{\mathbb{S}} := 4$.

($q_{\mathbb{S}}$ and $r_{\mathbb{S}}$ are prime.)

Let $\mathbb{S}_1^{(r)}$ be the subgroup of order $r_{\mathbb{S}}$ of the group of rational points on a Barreto–Lynn–Scott ([BLS2002]) curve $E_{\mathbb{S}_1}$ over $\mathbb{F}_{q_{\mathbb{S}}}$ with equation $y^2 = x^3 + b_{\mathbb{S}}$. This curve has embedding degree 12 with respect to $r_{\mathbb{S}}$.

Let $\mathbb{S}_2^{(r)}$ be the subgroup of order $r_{\mathbb{S}}$ in the sextic twist $E_{\mathbb{S}_2}$ of $E_{\mathbb{S}_1}$ over $\mathbb{F}_{q_{\mathbb{S}}{}^2}$ with equation $y^2 = x^3 + 4(i+1)$, where $i : \mathbb{F}_{q_{\mathbb{S}}{}^2}$.

We represent elements of $\mathbb{F}_{q_{\mathbb{S}}{}^2}$ as polynomials $a_1 \cdot t + a_0 : \mathbb{F}_{q_{\mathbb{S}}}[t]$, modulo the irreducible polynomial $t^2 + 1$; in this representation, i is given by t.

Let $\mathbb{S}_T^{(r)}$ be the subgroup of $r_{\mathbb{S}}{}^{\text{th}}$ roots of unity in $\mathbb{F}_{q_{\mathbb{S}}{}^{12}}^*$, with multiplicative identity $\mathbf{1}_{\mathbb{S}}$.

Let $\hat{e}_{\mathbb{S}}$ be the optimal ate pairing of type $\mathbb{S}_1^{(r)} \times \mathbb{S}_2^{(r)} \to \mathbb{S}_T^{(r)}$.

For $i : \{1 \,..\, 2\}$, let $\mathcal{O}_{\mathbb{S}_i}$ be the point at infinity in $\mathbb{S}_i^{(r)}$, and let $\mathbb{S}_i^{(r)*} := \mathbb{S}_i^{(r)} \setminus \{\mathcal{O}_{\mathbb{S}_i}\}$.

Let $\mathcal{P}_{\mathbb{S}_1} : \mathbb{S}_1^{(r)*} :=$

$(36854167537133870167810883151830777579616207957825464098945783786886075923783763188360549476763458215481041854645 07,$
$13395065449444764730204713799419212215849338759383496204265437364165114239563335064727246553533665349923917564415 69).$

Let $\mathcal{P}_{\mathbb{S}_2} : \mathbb{S}_2^{(r)*} :=$

$(30591443442442137099712598147537816369864703254766475586593732062916353247689584324335095631043470178378857633657 58 \cdot t +$
$35270106958746661818713911601106014489002995279277524021990864423979378573571502687334760034386517595276192630316 0,$
$92755366549233245574720196577603788075774019345359297002502797879397687700267556498094928972795756557543334421958 2 \cdot t +$
$19851506022872919355680545211771716383008689782156557308593786650663447263738237184238691042633339846414943403479 05).$

$\mathcal{P}_{\mathbb{S}_1}$ and $\mathcal{P}_{\mathbb{S}_2}$ are generators of $\mathbb{S}_1^{(r)}$ and $\mathbb{S}_2^{(r)}$ respectively.

Define $\mathsf{I2BEBSP} : (\ell : \mathbb{N}) \times \{0 \,..\, 2^\ell - 1\} \to \mathbb{B}^{[\ell]}$ as in Section 5.

For a point $P : \mathbb{S}_1^{(r)*} = (x_P, y_P)$:

- The field elements x_P and $y_P : \mathbb{F}_{q_\mathbb{S}}$ are represented as integers x and $y : \{0 \,..\, q_\mathbb{S} - 1\}$.
- Let $\tilde{y} = \begin{cases} 1, & \text{if } y > q_\mathbb{S} - y \\ 0, & \text{otherwise.} \end{cases}$
- P is encoded as $\mathsf{I2BEOSP}_{384}((4 + \tilde{y}) \cdot 2^{381} + x)$.

For a point $P : \mathbb{S}_2^{(r)*} = (x_P, y_P)$:

- Define $\mathsf{FE2IPP} : \mathbb{F}_{q_\mathbb{S}}[t]/(t^2 + 1) \to \{0 \,..\, q_\mathbb{S} - 1\}^{[2]}$ such that $\mathsf{FE2IPP}(a_{w,1} \cdot t + a_{w,0}) = [a_{w,1}, a_{w,0}]$.
- Let $x = \mathsf{FE2IPP}(x_P)$, $y = \mathsf{FE2IPP}(y_P)$, and $y' = \mathsf{FE2IPP}(-y_P)$.
- Let $\tilde{y} = \begin{cases} 1, & \text{if } y > y' \text{ lexicographically} \\ 0, & \text{otherwise.} \end{cases}$
- P is encoded as $\mathsf{I2BEOSP}_{384}((4 + \tilde{y}) \cdot 2^{381} + x_1) \,||\, \mathsf{I2BEOSP}_{384} x_2$.

Non-normative notes:

- Only the $r_\mathbb{S}$-order subgroups $\mathbb{S}_{1,2,T}^{(r)}$ are used, not their containing groups $\mathbb{S}_{1,2,T}$. Points in $\mathbb{S}_{1,2}^{(r)*}$ are *always* checked to be of order $r_\mathbb{S}$ when decoding from external representation. (Elements of $\mathbb{S}_T^{(r)}$ are never represented externally.) The (r) superscripts on $\mathbb{S}_{1,2,T}^{(r)}$ are used for consistency with notation elsewhere in this specification.
- The points at infinity $\mathcal{O}_{\mathbb{S}_{1,2}}$ never occur in proofs and have no defined encodings.
- In contrast to the corresponding BN-254 curve, $E_{\mathbb{S}_1}$ over $\mathbb{F}_{q_\mathbb{S}}$ is *not* of prime order.
- A rational point $P \neq \mathcal{O}_{\mathbb{S}_i}$ on the curve $E_{\mathbb{S}_i}$ for $i \in \{1, 2\}$ can be verified to be of order $r_\mathbb{S}$, and therefore in $\mathbb{S}_i^{(r)*}$, by checking that $r_\mathbb{S} \cdot P = \mathcal{O}_{\mathbb{S}_i}$.
- The encodings for $\mathbb{S}_{1,2}^{(r)*}$ are as used in Zcash.
- Algorithms for decompressing points from the encodings of $\mathbb{S}_{1,2}^{(r)*}$ are defined analogously to those for $\mathbb{G}_{1,2}^{(r)*}$ in Section 6.5.1, taking into account that the SORT compressed form (not the LSB compressed form) is used for $\mathbb{S}_1^{(r)*}$.

When computing square roots in $\mathbb{F}_{q_\mathbb{S}}$ or $\mathbb{F}_{q_\mathbb{S}^2}$ in order to decompress a point encoding, the implementation must not assume that the square root exists, or that the encoding represents a point on the curve.

6.5.3 Jubjub

"You boil it in sawdust: you salt it in glue:
You condense it with locusts and tape:
Still keeping one principal object in view—
To preserve its symmetrical shape."
— Lewis Carroll, "The Hunting of the Snark" [Carroll1876]

The elliptic curve Jubjub was designed in the context of the Zcash Sapling protocol, to be efficiently implementable in zk-SNARK circuits. The represented group $\mathbb{J}$ of points on this curve is defined in this section.

Let $q_{\mathbb{J}} := r_{\mathbb{S}}$, as defined in Section 6.5.2.

Let $r_{\mathbb{J}} := 6554484396890773809930967563523245729705921265872317281365359162392183254199$.

($q_{\mathbb{J}}$ and $r_{\mathbb{J}}$ are prime.)

Let $h_{\mathbb{J}} := 8$.

Let $a_{\mathbb{J}} := -1$.

Let $d_{\mathbb{J}} := -10240/10241 \pmod{q_{\mathbb{J}}}$.

Let $\mathbb{J}$ be the group of points (u, v) on a ctEdwards curve $E_{\mathbb{J}}$ over $\mathbb{F}_{q_{\mathbb{J}}}$ with equation $a_{\mathbb{J}} \cdot u^2 + v^2 = 1 + d_{\mathbb{J}} \cdot u^2 \cdot v^2$. The zero point with coordinates $(0, 1)$ is denoted $\mathcal{O}_{\mathbb{J}}$. $\mathbb{J}$ has order $h_{\mathbb{J}} \cdot r_{\mathbb{J}}$.

Let $\ell_{\mathbb{J}} := 256$.

Define $\mathsf{I2LEBSP} : (\ell : \mathbb{N}) \times \{0 \,..\, 2^\ell - 1\} \to \mathbb{B}^{[\ell]}$ as in Section 5.

Define $\mathsf{repr}_{\mathbb{J}} : \mathbb{J} \to \mathbb{B}^{[\ell_{\mathbb{J}}]}$ such that $\mathsf{repr}_{\mathbb{J}}(u, v) = \mathsf{I2LEBSP}_{256}(v + 2^{255} \cdot \tilde{u})$, where $\tilde{u} = u \bmod 2$.

Let $\mathsf{abst}_{\mathbb{J}} : \mathbb{B}^{[\ell_{\mathbb{J}}]} \to \mathbb{J} \cup \{\bot\}$ be the left inverse of $\mathsf{repr}_{\mathbb{J}}$ such that if S is not in the range of $\mathsf{repr}_{\mathbb{J}}$, then $\mathsf{abst}_{\mathbb{J}}(S) = \bot$.

Define $\mathbb{J}^{(r)}$ as the order-$r_{\mathbb{J}}$ subgroup of $\mathbb{J}$. Note that this includes $\mathcal{O}_{\mathbb{J}}$. For the set of points of order $r_{\mathbb{J}}$ (which excludes $\mathcal{O}_{\mathbb{J}}$), we write $\mathbb{J}^{(r)*}$.

Define $\mathbb{J}^{(r)}_\star := \{\mathsf{repr}_{\mathbb{J}}(P) : \mathbb{B}^{[\ell_{\mathbb{J}}]} \mid P \in \mathbb{J}^{(r)}\}$.

Non-normative notes:

- The ctEdwards compressed encoding used here is consistent with that used in EdDSA [BJLSY2015] for public keys and the R element of a signature.
- [BJLSY2015, "Encoding and parsing curve points"] gives algorithms for decompressing points from the encoding of $\mathbb{J}$.

When computing square roots in $\mathbb{F}_{q_{\mathbb{J}}}$ in order to decompress a point encoding, the implementation must not assume that the square root exists, or that the encoding represents a point on the curve.

This specification requires "strict" parsing as defined in [BJLSY2015, "Encoding and parsing integers"].

Note that algorithms elsewhere in this specification that use Jubjub may impose other conditions on points, for example that they have order at least $r_{\mathbb{J}}$.

6.5.4 Hash Extractor for ctEdwards curves

Let $\mathbb{G}$ be a ctEdwards curve over $\mathbb{F}_q$.

TODO: this was defined for Jubjub but should be applicable to any ctEdwards curve; double-check this.

TODO: this returns a bit sequence, but we might not want to convert to bits, especially in circuit contexts where that is expensive.

Let $\mathcal{U}((u, v)) = u$ and let $\mathcal{V}((u, v)) = v$.

Define $\mathsf{Extract}_{\mathbb{G}^{(r)}} : \mathbb{G}^{(r)} \to \mathbb{F}_q$ by

$\mathsf{Extract}_{\mathbb{G}^{(r)}}(P) := \mathsf{I2LEBSP}_{TODO}(\mathcal{U}(P))$.

Facts: The point $(0,1) = \mathcal{O}_{\mathbb{G}}$, and the point $(0,-1)$ has order 2 in $\mathbb{G}$. $\mathbb{G}^{(r)}$ is of odd-prime order.

Lemma 6.5.1. *Let $P = (u,v) \in \mathbb{G}^{(r)}$. Then $(u,-v) \notin \mathbb{G}^{(r)}$.*

Proof. If $P = \mathcal{O}_{\mathbb{G}}$ then $(u,-v) = (0,-1) \notin \mathbb{G}^{(r)}$. Else, P is of odd-prime order. Note that $v \neq 0$. (If $v = 0$ then $a \cdot u^2 = 1$, and so applying the doubling formula gives $[2]\,P = (0,-1)$, then $[4]\,P = (0,1) = \mathcal{O}_{\mathbb{G}}$; contradiction since then P would not be of odd-prime order.) Therefore, $-v \neq v$. Now suppose $(u,-v) = Q$ is a point in $\mathbb{G}^{(r)}$. Then by applying the doubling formula we have $[2]\,Q = -[2]\,P$. But also $[2]\,(-P) = -[2]\,P$. Therefore either $Q = -P$ (then $\mathcal{V}(Q) = \mathcal{V}(-P)$; contradiction since $-v \neq v$), or doubling is not injective on $\mathbb{G}^{(r)}$ (contradiction since $\mathbb{G}^{(r)}$ is of odd order [KvE2013]). □

Theorem 6.5.2. *$\mathcal{U}$ is injective on $\mathbb{G}^{(r)}$.*

Proof. By writing the curve equation as $v^2 = (1 - a \cdot u^2)/(1 - d \cdot u^2)$, and noting that the potentially exceptional case $1 - d \cdot u^2 = 0$ does not occur for a ctEdwards curve, we see that for a given u there can be at most two possible solutions for v, and that if there are two solutions they can be written as v and $-v$. In that case by the Lemma, at most one of (u,v) and $(u,-v)$ is in $\mathbb{G}^{(r)}$. Therefore, $\mathcal{U}$ is injective on points in $\mathbb{G}^{(r)}$. □

Since $\mathsf{I2LEBSP}_{TODO}$ is injective, it follows that $\mathsf{Extract}_{\mathbb{G}^{(r)}}$ is injective on $\mathbb{G}^{(r)}$.

6.5.5 Group Hash into a ctEdwards curve

Let $\mathbb{G}$ be a ctEdwards curve over $\mathbb{F}_q$, and let $\mathbb{G}^{(r)}$, $\mathbb{G}^{(r)*}$, and $\mathsf{abst}_{\mathbb{G}}$ be as defined in Section 3.8.

TODO: this was defined for Jubjub but should be applicable to any ctEdwards curve; double-check this.

TODO: allow use of a general hash (suitable for a random oracle) other than BLAKE2s-256.

Let $\mathsf{GroupHash.Input} := \mathbb{B}^{\mathbb{Y}^{[8]}} \times \mathbb{B}^{\mathbb{Y}^{[\mathbb{N}]}}$, and let $\mathsf{GroupHash.URSType} := \mathbb{B}^{\mathbb{Y}^{[64]}}$.

(The input element with type $\mathbb{B}^{\mathbb{Y}^{[8]}}$ is intended to act as a "personalization" parameter to distinguish uses of the group hash for different purposes.)

TODO: define URS without relying on the Sapling randomness beacon.

Let BLAKE2s-256 be as defined in Section 6.1.1.

Let LEOS2IP be as defined in Section 5.

Let $D : \mathbb{B}^{\mathbb{Y}^{[8]}}$ be an 8-byte domain separator, and let $M : \mathbb{B}^{\mathbb{Y}^{[\mathbb{N}]}}$ be the hash input.

The hash $\mathsf{GroupHash}^{\mathbb{G}^{(r)}}_{\mathsf{URS}}(D, M) : \mathbb{G}^{(r)*}$ is calculated as follows:

let $\underline{H} = \mathsf{BLAKE2s\text{-}256}(D, \mathsf{URS} \,||\, M)$
let $P = \mathsf{abst}_{\mathbb{G}_(}\mathsf{LEOS2BSP}_{256}(\underline{H}))$
if $P = \perp$ then return $\perp$
let $Q = [h_{\mathbb{G}}]\,P$
if $Q = \mathcal{O}_{\mathbb{G}}$ then return $\perp$, else return Q.

Notes:

- The BLAKE2s-256 chaining variable after processing URS may be precomputed.

- The use of $\mathsf{GroupHash}^{\mathbb{G}^{(r)}}_{\mathsf{URS}}$ to generate independent bases needs a random oracle (with domain given by inputs on which $\mathsf{GroupHash}^{\mathbb{G}^{(r)}}_{\mathsf{URS}}$ does not return $\bot$); here we show that it is sufficient to employ a simpler random oracle instantiated by BLAKE2s-256 in the security analysis.

  $\underline{H} : \mathbb{B}^{\mathbb{Y}[32]} \mapsto_{\neq\bot} \mathsf{abst}_{\mathbb{G}_(}\mathsf{LEOS2BSP}_{256}(\underline{H})) : \mathbb{G}$ is injective, and both it and its inverse are efficiently computable.

  $P : \mathbb{G} \mapsto_{\neq\mathcal{O}_\mathbb{G}} [h_\mathbb{G}]\,P : \mathbb{G}^{(r)*}$ is exactly $h_\mathbb{G}$-to-1, and both it and its inverse relation are efficiently computable.

  It follows that when $\big(D : \mathbb{B}^{\mathbb{Y}[8]}, M : \mathbb{B}^{\mathbb{Y}[\mathbb{N}]}\big) \mapsto \mathsf{BLAKE2s\text{-}256}(D, \mathsf{URS} \,||\, M) : \mathbb{B}^{\mathbb{Y}[32]}$ is modelled as a random oracle, $\big(D : \mathbb{B}^{\mathbb{Y}[8]}, M : \mathbb{B}^{\mathbb{Y}[\mathbb{N}]}\big) \mapsto_{\neq\bot} \mathsf{GroupHash}^{\mathbb{G}^{(r)}}_{\mathsf{URS}}\big(D, M\big) : \mathbb{G}^{(r)*}$ also acts as a random oracle.

Define $\mathsf{first} : (\mathbb{B}^{\mathbb{Y}} \to T \cup \{\bot\}) \to T \cup \{\bot\}$ so that $\mathsf{first}(f) = f(i)$ where i is the least integer in $\mathbb{B}^{\mathbb{Y}}$ such that $f(i) \neq \bot$, or $\bot$ if no such i exists.

Define $\mathsf{FindGroupHash}^{\mathbb{G}^{(r)*}}\big(D, M\big) := \mathsf{first}(i : \mathbb{B}^{\mathbb{Y}} \mapsto \mathsf{GroupHash}^{\mathbb{G}^{(r)}}_{\mathsf{URS}}(D, M \,||[i]) : \mathbb{G}^{(r)*} \cup \{\bot\})$.

Note: For random input, $\mathsf{FindGroupHash}^{\mathbb{G}^{(r)*}}$ returns $\bot$ with probability approximately 2^{-256}.

7 Acknowledgements

Portions of this document were extracted from the Zcash Protocol Specification by Daira Hopwood.

8 References

[ABR1999] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. *DHAES: An Encryption Scheme Based on the Diffie-Hellman Problem*. Cryptology ePrint Archive: Report 1999/007. Received March 17, 1999. September 1998. URL: `https://eprint.iacr.org/1999/007` (visited on 2016-08-21) (↑ p 4).

[AKLGL2010] Diego Aranha, Koray Karabina, Patrick Longa, Catherine Gebotys, and Julio López. *Faster Explicit Formulas for Computing Pairings over Ordinary Curves*. Cryptology ePrint Archive: Report 2010/526. Last revised September 12, 2011. URL: `https://eprint.iacr.org/2010/526` (visited on 2018-04-03) (↑ p 14).

[ANWW2013] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. *BLAKE2: simpler, smaller, fast as MD5*. January 29, 2013. URL: `https://blake2.net/#sp` (visited on 2016-08-14) (↑ p 10).

[BBJLP2008] Daniel Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. *Twisted Edwards Curves*. Cryptology ePrint Archive: Report 2008/013. Received January 8, 2008. March 13, 2008. URL: `https://eprint.iacr.org/2008/013` (visited on 2018-01-12) (↑ p 26, 27).

[BCP1988] Jurgen Bos, David Chaum, and George Purdy. "A Voting Scheme". Unpublished. Presented at the rump session of CRYPTO '88 (Santa Barbara, California, USA, August 21–25, 1988); does not appear in the proceedings. (↑ p 10).

[BDLSY2012] Daniel Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-speed high-security signatures". In: *Journal of Cryptographic Engineering* 2 (September 26, 2011), pages 77–89. URL: `http://cr.yp.to/papers.html#ed25519` (visited on 2016-08-14). Document ID: a1a62a2f76d23f65d622484ddd09caf8. (↑ p 32).

[Bernstein2001] Daniel Bernstein. *Pippenger's exponentiation algorithm*. December 18, 2001. URL: `https://cr.yp.to/papers.html#pippenger` (visited on 2018-07-27). Draft. To be incorporated into the author's *High-speed cryptography* book. Error pointed out by Sam Hocevar: the example in Figure 4 needs 2 and is thus of length 18. (↑ p 32, 33).

[Bernstein2006] Daniel Bernstein. "Curve25519: new Diffie-Hellman speed records". In: *Public Key Cryptography – PKC 2006. Proceedings of the 9th International Conference on Theory and Practice in Public-Key Cryptography (New York, NY, USA, April 24–26, 2006)*. Springer-Verlag, February 9, 2006. URL: http://cr.yp.to/papers.html#curve25519 (visited on 2016-08-14). Document ID: 4230efdfa673480fc079449d90f322c0. (↑ p 4).

[BGG1995] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. "Incremental Cryptography: The Case of Hashing and Signing". In: *Advances in Cryptology - CRYPTO '94. Proceedings of the 14th Annual International Cryptology Conference (Santa Barbara, California, USA, August 21–25, 1994)*. Ed. by Yvo Desmedt. Vol. 839. Lecture Notes in Computer Science. Springer, October 20, 1995, pages 216–233. ISBN: 978-3-540-48658-9. DOI: 10.1007/3-540-48658-5_22. URL: https://cseweb.ucsd.edu/~mihir/papers/inc1.pdf (visited on 2018-02-09) (↑ p 10, 11, 30).

[BJLSY2015] Daniel Bernstein, Simon Josefsson, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. *EdDSA for more curves*. Technical Report. July 4, 2015. URL: https://cr.yp.to/papers.html#eddsa (visited on 2018-01-22) (↑ p 11, 17).

[BL2017] Daniel Bernstein and Tanja Lange. *Montgomery curves and the Montgomery ladder*. Cryptology ePrint Archive: Report 2017/293. Received March 30, 2017. URL: https://eprint.iacr.org/2017/293 (visited on 2017-11-26) (↑ p 8, 26, 27).

[BLS2002] Paulo Barreto, Ben Lynn, and Michael Scott. *Constructing Elliptic Curves with Prescribed Embedding Degrees*. Cryptology ePrint Archive: Report 2002/088. Last revised February 22, 2005. URL: https://eprint.iacr.org/2002/088 (visited on 2018-04-20) (↑ p 15).

[BN2005] Paulo Barreto and Michael Naehrig. *Pairing-Friendly Elliptic Curves of Prime Order*. Cryptology ePrint Archive: Report 2005/133. Last revised February 28, 2006. URL: https://eprint.iacr.org/2005/133 (visited on 2018-04-20) (↑ p 14).

[BN2007] Mihir Bellare and Chanathip Namprempre. *Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm*. Cryptology ePrint Archive: Report 2000/025. Last revised July 14, 2007. URL: https://eprint.iacr.org/2000/025 (visited on 2016-09-02) (↑ p 4).

[Bowe2017] Sean Bowe. *ebfull/pairing source code, BLS12-381 – README.md as of commit e726600*. URL: https://github.com/ebfull/pairing/tree/e72660056e00c93d6b054dfb08ff34a1c67cb799/src/bls12_381 (visited on 2017-07-16) (↑ p 15).

[Carroll1876] Lewis Carroll. *The Hunting of the Snark*. With illustrations by Henry Holiday. MacMillan and Co. London. March 29, 1876. URL: https://www.gutenberg.org/files/29888/29888-h/29888-h.htm (visited on 2018-05-23) (↑ p 17).

[CDvdG1987] David Chaum, Ivan Damgård, and Jeroen van de Graaf. "Multiparty computations ensuring privacy of each party's input and correctness of the result". In: *Advances in Cryptology - CRYPTO '87. Proceedings of the 14th Annual International Cryptology Conference (Santa Barbara, California, USA, August 16–20, 1987)*. Ed. by Carl Pomerance. Vol. 293. Lecture Notes in Computer Science. Springer, January 1988, pages 87–119. ISBN: 978-3-540-48184-3. DOI: 10.1007/3-540-48184-2_7. URL: https://www.researchgate.net/profile/Jeroen_Van_de_Graaf/publication/242379939_Multiparty_computations_ensuring_secrecy_of_each_party%27s_input_and_correctness_of_the_output (visited on 2018-03-01) (↑ p 10).

[CvHP1991] David Chaum, Eugène van Heijst, and Birgit Pfitzmann. *Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer*. February 1991. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.8570 (visited on 2018-02-17). An extended abstract appeared in *Advances in Cryptology - CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference (Santa Barbara, California, USA, August 11–15, 1991)*; Ed. by Joan Feigenbaum; Vol. 576, Lecture Notes in Computer Science, pages 470–484; Springer, 1992; ISBN 978-3-540-55188-1. (↑ p 10, 30).

[deRooij1995] Peter de Rooij. "Efficient exponentiation using precomputation and vector addition chains". In: *Advances in Cryptology - EUROCRYPT '94. Proceedings, Workshop on the Theory and Application of Cryptographic Techniques (Perugia, Italy, May 9–12, 1994)*. Ed. by Alfredo De Santis. Vol. 950. Lecture Notes in Computer Science. Springer, pages 389–399. ISBN: 978-3-540-60176-0. DOI: 10.1007/BFb0053453. URL: https://link.springer.com/chapter/10.1007/BFb0053453 (visited on 2018-07-27) (↑ p 32, 33).

[DS2016] David Derler and Daniel Slamanig. *Key-Homomorphic Signatures and Applications to Multiparty Signatures and Non-Interactive Zero-Knowledge*. Cryptology ePrint Archive: Report 2016/792. Last revised February 6, 2017. URL: https://eprint.iacr.org/2016/792 (visited on 2018-04-09) (↑ p 6).

[FKMSSS2016] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. *Efficient Unlinkable Sanitizable Signatures from Signatures with Re-Randomizable Keys*. Cryptology ePrint Archive: Report 2012/159. Last revised February 11, 2016. URL: https://eprint.iacr.org/2015/395 (visited on 2018-03-03). An extended abstract appeared in *Public Key Cryptography – PKC 2016: 19th IACR International Conference on Practice and Theory in Public-Key Cryptography (Taipei, Taiwan, March 6–9, 2016), Proceedings, Part 1*; Ed. by Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang; Vol. 9614, Lecture Notes in Computer Science, pages 301–330; Springer, 2016; ISBN 978-3-662-49384-7. (↑ p 5, 6, 11).

[Groth2016] Jens Groth. *On the Size of Pairing-based Non-interactive Arguments*. Cryptology ePrint Archive: Report 2016/260. Last revised May 31, 2016. URL: https://eprint.iacr.org/2016/260 (visited on 2017-08-03) (↑ p 33).

[IEEE2000] IEEE Computer Society. *IEEE Std 1363-2000: Standard Specifications for Public-Key Cryptography*. IEEE, August 29, 2000. DOI: 10.1109/IEEESTD.2000.92292. URL: http://ieeexplore.ieee.org/servlet/opac?punumber=7168 (visited on 2016-08-03) (↑ p 15).

[IEEE2004] IEEE Computer Society. *IEEE Std 1363a-2004: Standard Specifications for Public-Key Cryptography – Amendment 1: Additional Techniques*. IEEE, September 2, 2004. DOI: 10.1109/IEEESTD.2004.94612. URL: http://ieeexplore.ieee.org/servlet/opac?punumber=9276 (visited on 2016-08-03) (↑ p 15).

[KvE2013] Kaa1el and Hagen von Eitzen. *If a group G has odd order, then the square function is injective (answer)*. Mathematics Stack Exchange. URL: https://math.stackexchange.com/a/522277/185422 (visited on 2018-02-08). Version: 2013-10-11. (↑ p 18).

[SVPBABW2012] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Muqeet Ali, Andrew J. Blumberg, and Michael Walfish. *Taking proof-based verified computation a few steps closer to practicality (extended version)*. Cryptology ePrint Archive: Report 2012/598. Last revised February 28, 2013. URL: https://eprint.iacr.org/2012/598.pdf (visited on 2018-04-25) (↑ p 23).

[Vercauter2009] Frederik Vercauteren. *Optimal pairings*. Cryptology ePrint Archive: Report 2008/096. Last revised March 7, 2008. URL: https://eprint.iacr.org/2008/096 (visited on 2018-04-06). A version of this paper appeared in *IEEE Transactions of Information Theory*, Vol. 56, pages 455–461; IEEE, 2009. (↑ p 14).

Part I

Appendices

A Circuit Design

A.1 Constraint Programs

A circuit is defined in terms of a constraint program specifying a *Rank-1 Constraint System* (R1CS), as detailed in this section.

Let $\mathbb{F}$ be a finite field.

A constraint program consists of a set of constraints over variables in $\mathbb{F}$, each of the form:

$$\big(A\big) \times \big(B\big) = \big(C\big)$$

where $\big(A\big)$, $\big(B\big)$, and $\big(C\big)$ are linear combinations of variables and constants in $\mathbb{F}$.

Here $\times$ and $\cdot$ both represent multiplication in the field $\mathbb{F}$, but we use $\times$ for multiplications corresponding to gates of the circuit, and $\cdot$ for multiplications by constants in the terms of a linear combination. $\times$ should not be confused with $\times$ which is defined as cartesian product in Section 2.

B Circuit Components

Each of the following sections describes how to implement a particular circuit component, and counts the number of constraints required.

It is important for security to ensure that variables intended to be of boolean type are boolean-constrained; and for efficiency that they are boolean-constrained only once. We explicitly state for the boolean inputs and outputs of each component whether they are boolean-constrained by the component, or are assumed to have been boolean-constrained separately.

Affine coordinates for elliptic curve points are assumed to represent points on the relevant curve, unless otherwise specified.

In this section, variables have type $\mathbb{F}$ unless otherwise specified. In contrast to most of this document, we use zero-based indexing in order to more closely match the implementation.

B.1 Operations on individual bits

B.1.1 Boolean constraints

A boolean constraint $b \in \mathbb{B}$ can be implemented as:

$$\big(1 - b\big) \times \big(b\big) = \big(0\big)$$

B.1.2 Conditional equality

The constraint "either $a = 0$ or $b = c$" can be implemented as:

$$\big(a\big) \times \big(b - c\big) = \big(0\big)$$

B.1.3 Selection constraints

A selection constraint $(b \;?\; x : y) = z$, where $b : \mathbb{B}$ has been boolean-constrained, can be implemented as:

$$\big(b\big) \;\mathsf{x}\; \big(y - x\big) \;=\; \big(y - z\big)$$

B.1.4 Nonzero constraints

Since only nonzero elements of $\mathbb{F}$ have a multiplicative inverse, the assertion $a \neq 0$ can be implemented by witnessing the inverse, $a_{\mathsf{inv}} = a^{-1} \pmod{}$:

$$\big(a_{\mathsf{inv}}\big) \;\mathsf{x}\; \big(a\big) \;=\; \big(1\big)$$

This technique comes from [SVPBABW2012, Appendix D.1].

Non-normative note: A global optimization allows to use a single inverse computation outside the circuit for any number of nonzero constraints. Suppose that we have n variables (or linear combinations) that are supposed to be nonzero: $a_{0\,..\,n-1}$. Multiply these together (using $n-1$ constraints) to give $a^* = \prod_{i=0}^{n-1} a_i$; then, constrain a^* to be nonzero. This works because the product a^* is nonzero if and only if all of $a_{0\,..\,n-1}$ are nonzero.

B.1.5 Exclusive-or constraints

An exclusive-or operation $a \oplus b = c$, where $a, b : \mathbb{B}$ are already boolean-constrained, can be implemented in one constraint as:

$$\big(2 \cdot a\big) \;\mathsf{x}\; \big(b\big) \;=\; \big(a + b - c\big)$$

This automatically boolean-constrains c. Its correctness can be seen by checking the truth table of (a, b).

B.2 Operations on multiple bits

B.2.1 [Un]packing

Let $n : \mathbb{N}^+$ be a constant. The operation of converting a field element, $a : \mathbb{F}_q$, to a sequence of boolean variables $b_{0\,..\,n-1} : \mathbb{B}^{[n]}$ such that $a = \sum_{i=0}^{n-1} b_i \cdot 2^i \pmod{q}$, is called "unpacking". The inverse operation is called "packing".

In the constraint program these are the same operation (but see the note about canonical representation below). We assume that the variables $b_{0\,..\,n-1}$ are boolean-constrained separately.

We have $a \bmod q = \left(\sum_{i=0}^{n-1} b_i \cdot 2^i\right) \bmod q = \left(\sum_{i=0}^{n-1} b_i \cdot (2^i \bmod q)\right) \bmod q$.

This can be implemented in one constraint:

$$\left(\sum_{i=0}^{n-1} b_i \cdot (2^i \bmod r_{\mathbb{S}})\right) \;\mathsf{x}\; \big(1\big) \;=\; \big(a\big)$$

Notes:

- The bit length n is not limited by the field element size.
- Since the constraint has only a trivial multiplication, it is possible to eliminate it by merging it into the boolean constraint of one of the output bits, expressing that bit as a linear combination of the others and a.
- If $2^n > q$, there may be multiple representations of $a : \mathbb{F}_q$ as a sequence of n bits. This is a potential hazard, but it may or may not be necessary to force use of the canonical representation $\mathsf{I2LEBSP}_n(a)$, depending on the context in which the [un]packing operation is used. We therefore do not consider this to be part of the [un]packing operation itself.

B.2.2 Range check

Let $n : \mathbb{N}^+$ be a constant, and let $a = \sum_{i=0}^{n-1} a_i \cdot 2^i : \mathbb{N}$. Suppose we want to constrain $a \leq c$ for some *constant* $c = \sum_{i=0}^{n-1} c_i \cdot 2^i : \mathbb{N}$.

Without loss of generality we can assume that $c_{n-1} = 1$, because if it were not then we would decrease n accordingly.

Note that since a and c are provided in binary representation, their bit length n is not limited by the field element size. We *do not* assume that the bits $a_{0\,..\,n-1}$ are already boolean-constrained.

Define $\Pi_m = \prod_{i=m}^{n-1} (c_i = 0 \vee a_i = 1)$ for $m \in \{0\,..\,n-1\}$. Notice that for any $m < n-1$ such that $c_m = 0$, we have $\Pi_m = \Pi_{m+1}$, and so it is only necessary to allocate separate variables for the Π_m such that $m < n-1$ and $c_m = 1$. Furthermore if $c_{n-2\,..\,0}$ has $t > 0$ trailing 1 bits, then we do not need to allocate variables for $\Pi_{0\,..\,t-1}$ because those variables will not be used below.

More explicitly:

Let $\Pi_{n-1} = a_{n-1}$.

For i from $n-2$ down to t,

- if $c_i = 0$, then let $\Pi_i = \Pi_{i+1}$;
- if $c_i = 1$, then constrain $\big(\Pi_{i+1}\big)$ `x` $\big(a_i\big)$ = $\big(\Pi_i\big)$.

Then we constrain the a_i as follows:

For i from $n-1$ down to 0,

- if $c_i = 0$, constrain $\big(1 - \Pi_{i+1} - a_i\big)$ `x` $\big(a_i\big)$ = $\big(0\big)$;
- if $c_i = 1$, boolean-constrain a_i as in Section B.1.1.

Note that the constraints corresponding to zero bits of c are *in place of* boolean constraints on bits of a_i.

This costs $n + k$ constraints, where k is the number of non-trailing 1 bits in $c_{n-2\,..\,0}$.

Theorem B.2.1 (Correctness of a constraint system for range checks). *Assume $c_{0\,..\,n-1} : \mathbb{B}^{[n]}$ and $c_{n-1} = 1$. Define $A_m := \sum_{i=m}^{n-1} a_i \cdot 2^i$ and $C_m := \sum_{i=m}^{n-1} c_i \cdot 2^i$. For any $m \in \{0\,..\,n-1\}$, $A_m \leq C_m$ iff the restriction of the above constraint system to $i \in \{m\,..\,n-1\}$ is satisfied. Furthermore the system at least boolean-constrains $a_{0\,..\,n-1}$.*

Proof. For $i \in \{0\,..\,n-1\}$ such that $c_i = 1$, the corresponding a_i are unconditionally boolean-constrained. This implies that the system constrains $\Pi_i \in \mathbb{B}$ for all $i \in \{0\,..\,n-1\}$. For $i \in \{0\,..\,n-1\}$ such that $c_i = 0$, the constraint $\big(1 - \Pi_{i+1} - a_i\big)$ `x` $\big(a_i\big)$ = $\big(0\big)$ constrains a_i to be 0 if $\Pi_{i+1} = 1$, otherwise it constrains $a_i \in \mathbb{B}$. So all of $a_{0\,..\,n-1}$ are at least boolean-constrained.

To prove the rest of the theorem we proceed by induction on decreasing m, i.e. taking successively longer prefixes of the big-endian binary representations of a and c.

Base case $m = n-1$: since $c_{n-1} = 1$, the constraint system has just one boolean constraint on a_{n-1}, which fulfils the theorem since $A_{n-1} \leq C_{n-1}$ is always satisfied.

Inductive case $m < n-1$:

- If $A_{m+1} > C_{m+1}$, then by the inductive hypothesis the constraint system must fail, which fulfils the theorem regardless of the value of a_m.
- If $A_{m+1} \leq C_{m+1}$, then by the inductive hypothesis the constraint system restricted to $i \in \{m+1\,..\,n-1\}$ succeeds. We have $\Pi_{m+1} = \prod_{i=m+1}^{n-1} (c_i = 0 \vee a_i = 1) = \prod_{i=m+1}^{n-1} (a_i \geq c_i)$.
  - If $A_{m+1} = C_{m+1}$, then $a_i = c_i$ for all $i \in \{m+1\,..\,n-1\}$ and so $\Pi_{m+1} = 1$. Also $A_m \leq C_m$ iff $a_m \leq c_m$.
    When $c_m = 1$, only a boolean constraint is added for a_m which fulfils the theorem.
    When $c_m = 0$, a_m is constrained to be 0 which fulfils the theorem.

– If $A_{m+1} < C_{m+1}$, then it cannot be the case that $a_i \geq c_i$ for all $i \in \{m+1 \,..\, n-1\}$, so $\Pi_{m+1} = 0$. This implies that the constraint on a_m is always equivalent to a boolean constraint, which fulfils the theorem because $A_m \leq C_m$ must be true regardless of the value of a_m.

This covers all cases. □

Correctness of the full constraint system follows by taking $m = 0$ in the above theorem.

The algorithm in Section B.3.2 uses range checks with $c = r_{\mathbb{S}} - 1$ to validate ctEdwards compressed encodings.

Non-normative note: It is possible to optimize the computation of $\Pi_{t \,..\, n-2}$ further. Notice that Π_m is only used when m is the index of the last bit of a run of 1 bits in c. So for each such run of 1 bits $c_{m \,..\, m+N-2}$ of length $N-1$, it is sufficient to compute an N-ary AND of $a_{m \,..\, m+N-2}$ and Π_{m+N-1}: $R = \prod_{i=0}^{N-1} X_i$. This can be computed in 3 constraints for any N; boolean-constrain the output R, and then add constraints

$$\left(N - \sum_{i=0}^{N-1} X_i\right) \;\mathsf{x}\; (\mathsf{inv}) \;=\; (1 - R) \quad \text{to enforce that } \sum_{i=0}^{N-1} X_i \neq N \text{ when } R = 0;$$

$$\left(N - \sum_{i=0}^{N-1} X_i\right) \;\mathsf{x}\; (R) \;=\; (0) \qquad \text{to enforce that } \sum_{i=0}^{N-1} X_i = N \text{ when } R = 1.$$

where inv is witnessed as $(N - \sum_{i=0}^{N-1} X_i)^{-1}$ if $R = 0$ or is unconstrained otherwise. (Since $N < r_{\mathbb{S}}$, the sums cannot overflow.)

In fact the last constraint is not needed in this context because it is sufficient to compute an upper bound on each Π_m (i.e. it does not benefit a malicious prover to witness $R = 1$ when the result of the AND should be 0). So the cost of computing Π variables for an arbitrarily long run of 1 bits can be reduced to 2 constraints.

B.3 Elliptic curve operations

B.3.1 Checking that affine ctEdwards coordinates are on the curve

To check that (u, v) is a point on the ctEdwards curve, we can use:

$$(u) \;\mathsf{x}\; (u) \;=\; (uu)$$
$$(v) \;\mathsf{x}\; (v) \;=\; (vv)$$
$$(d_{\mathbb{J}} \cdot uu) \;\mathsf{x}\; (vv) \;=\; (a_{\mathbb{J}} \cdot uu + vv - 1).$$

B.3.2 ctEdwards [de]compression and validation

Define $\mathsf{DecompressValidate} : \mathsf{CompressedCtEdwardsJubjub} \to \mathsf{AffineCtEdwardsJubjub}$ as follows:

$\mathsf{DecompressValidate}(\tilde{u}, v)$:

// Prover supplies the u-coordinate.

Let $u : \mathbb{F}$.

// Section B.3.1.

Check that (u, v) is a point on the ctEdwards curve.

// Section B.2.1.

Unpack u to $\sum_{i=0}^{254} u_i \cdot 2^i$, equating $\tilde{u}$ with u_0.

// Section B.2.2.

Check that $\sum_{i=0}^{254} u_i \cdot 2^i \leq r_{\mathbb{S}} - 1$.

Return (u, v).

The same constraint program is used for compression and decompression.

Non-normative note: The point-on-curve check could be omitted if (u, v) were already known to be on the curve. However, keeping it in this case provides a consistency check on the elliptic curve arithmetic.

B.3.3 ctEdwards ↔ Montgomery conversion

TODO: this was defined for Jubjub; generalize to arbitrary ctEdwards/Montgomery conversions.

Let c be a constant that depends on the curve; for Jubjub, let $c = \sqrt[+]{-40964}$.

Define $\mathsf{CtEdwardsToMontgomery} : \mathsf{AffineCtEdwards} \to \mathsf{AffineMontgomery}$ as follows:

$$\mathsf{CtEdwardsToMont}(u, v) = \left(\tfrac{1+v}{1-v},\ c \cdot \tfrac{1+v}{(1-v)\cdot u}\right) \qquad [1 - v \neq 0 \ \text{ and } \ u \neq 0]$$

Define $\mathsf{MontgomeryToCtEdwards} : \mathsf{AffineMontgomery} \to \mathsf{AffineCtEdwards}$ as follows:

$$\mathsf{MontToCtEdwards}(x, y) = \left(c \cdot \tfrac{x}{y},\ \tfrac{x-1}{x+1}\right) \qquad [x + 1 \neq 0 \ \text{ and } \ y \neq 0]$$

Either of these conversions can be implemented by the same constraint program:

$$(y) \ \mathsf{x} \ (u) \ = \ (c \cdot x)$$
$$(x + 1) \ \mathsf{x} \ (v) \ = \ (x - 1)$$

The above conversions should only be used if the input is guaranteed to be a point on the relevant curve. If that is the case, the theorems below enumerate all exceptional inputs that may violate the side-conditions.

Theorem B.3.1 (Exceptional points (ctEdwards → Montgomery)). *Let (u, v) be an affine point on a ctEdwards curve $E_{\mathsf{ctEdwards}(a,d)}$. Then the only points with $u = 0$ or $1 - v = 0$ are $(0, 1) = \mathcal{O}_{\mathbb{J}}$, and $(0, -1)$ of order 2.*

Proof. The curve equation is $a \cdot u^2 + v^2 = 1 + d \cdot u^2 \cdot v^2$ with $a \neq d$ (see [BBJLP2008, Definition 2.1]). By substituting $u = 0$ we obtain $v = \pm 1$, and by substituting $v = 1$ and using $a \neq d$ we obtain $u = 0$. □

Theorem B.3.2 (Exceptional points (Montgomery → ctEdwards)). *Let (x, y) be an affine point on a Montgomery curve $E_{\mathsf{Montgomery}(A,B)}$ over $\mathbb{F}_r$ with parameters A and B such that $A^2 - 4$ is a nonsquare in $\mathbb{F}_r$, that is birationally equivalent to a ctEdwards curve. Then $x + 1 \neq 0$, and the only point (x, y) with $y = 0$ is $(0, 0)$ of order 2.*

Proof. That the only point with $y = 0$ is $(0, 0)$ is proven above.

If $x + 1 = 0$, then subtituting $x = -1$ into the Montgomery curve equation gives $B \cdot y^2 = x^3 + A \cdot x^2 + x = A - 2$. So in that case $y^2 = (A-2)/B$. The right-hand-side is equal to the parameter d of a particular ctEdwards curve birationally equivalent to the Montgomery curve (see [BL2017, section 4.3.5]). For all ctEdwards curves, d is nonsquare, so this equation has no solutions for y, hence $x + 1 \neq 0$. □

B.3.4 Affine Montgomery arithmetic

The incomplete affine Montgomery addition formulae given in [BL2017, section 4.3.2] are:

$$x_3 = B_{\mathbb{M}} \cdot \lambda^2 - A_{\mathbb{M}} - x_1 - x_2$$
$$y_3 = (x_1 - x_3) \cdot \lambda - y_1$$
$$\text{where } \lambda = \begin{cases} \frac{3 \cdot x_1^2 + 2 \cdot A_{\mathbb{M}} \cdot x_1 + 1}{2 \cdot B_{\mathbb{M}} \cdot y_1}, & \text{if } x_1 = x_2 \\ \frac{y_2 - y_1}{x_2 - x_1}, & \text{otherwise.} \end{cases}$$

The following theorem helps to determine when these incomplete addition formulae can be safely used [TODO generalize this to also cover short Weierstrass (as in the Halo paper)]:

Theorem B.3.3 (Distinct-x theorem). *Let Q be a point of odd-prime order s on a Montgomery curve $\mathbb{M} = E_{\mathsf{Montgomery}(A_{\mathbb{M}},B_{\mathbb{M}})}$ over $\mathbb{F}$. Let $k_{1\,..\,2}$ be integers in $\left\{-\frac{s-1}{2} \,..\, \frac{s-1}{2}\right\} \setminus \{0\}$. Let $P_i = [k_i]\,Q = (x_i, y_i)$ for $i \in \{1\,..\,2\}$, with $k_2 \neq \pm k_1$. Then the non-unified addition constraints*

$$\begin{aligned}
(x_2 - x_1) \;\mathsf{x}\; (\lambda) &= (y_2 - y_1) \\
(B_{\mathbb{M}} \cdot \lambda) \;\mathsf{x}\; (\lambda) &= (A_{\mathbb{M}} + x_1 + x_2 + x_3) \\
(x_1 - x_3) \;\mathsf{x}\; (\lambda) &= (y_3 + y_1)
\end{aligned}$$

implement the affine Montgomery addition $P_1 + P_2 = (x_3, y_3)$ for all such $P_{1\,..\,2}$.

Proof. The given constraints are equivalent to the Montgomery addition formulae under the side condition that $x_1 \neq x_2$. (Note that neither P_i can be the zero point since $k_{1\,..\,2} \neq 0 \pmod{s}$.) Assume for a contradiction that $x_1 = x_2$. For any $P_1 = [k_1]\,Q$, there can be only one other point $-P_1$ with the same x-coordinate. (This follows from the fact that the curve equation determines $\pm y$ as a function of x.) But $-P_1 = [-1]\,[k_1]\,Q = [-k_1]\,Q$. Since $k : \left\{-\frac{s-1}{2} \,..\, \frac{s-1}{2}\right\} \mapsto [k]\,Q : \mathbb{M}$ is injective and $k_{1\,..\,2}$ are in $\left\{-\frac{s-1}{2} \,..\, \frac{s-1}{2}\right\}$, then $k_2 = \pm k_1$ (contradiction). □

The conditions of this theorem are called the distinct-x criterion.

In particular, if $k_{1\,..\,2}$ are integers in $\left\{1 \,..\, \frac{s-1}{2}\right\}$ then it is sufficient to require $k_2 \neq k_1$, since that implies $k_2 \neq \pm k_1$.

Affine Montgomery doubling can be implemented as:

$$\begin{aligned}
(x) \;\mathsf{x}\; (x) &= (xx) \\
(2 \cdot B_{\mathbb{M}} \cdot y) \;\mathsf{x}\; (\lambda) &= (3 \cdot xx + 2 \cdot A_{\mathbb{M}} \cdot x + 1) \\
(B_{\mathbb{M}} \cdot \lambda) \;\mathsf{x}\; (\lambda) &= (A_{\mathbb{M}} + 2 \cdot x + x_3) \\
(x - x_3) \;\mathsf{x}\; (\lambda) &= (y_3 + y)
\end{aligned}$$

This doubling formula is valid when $y \neq 0$, which is the case when (x, y) is not the point $(0, 0)$ (the only point of order 2).

B.3.5 Affine ctEdwards arithmetic

Formulae for affine ctEdwards addition are given in [BBJLP2008, section 6]. With a change of variable names to match our convention, the formulae for $(u_1, v_1) + (u_2, v_2) = (u_3, v_3)$ are:

$$u_3 = \frac{u_1 \cdot v_2 + v_1 \cdot u_2}{1 + d_{\mathbb{J}} \cdot u_1 \cdot u_2 \cdot v_1 \cdot v_2}$$

$$v_3 = \frac{v_1 \cdot v_2 - a_{\mathbb{J}} \cdot u_1 \cdot u_2}{1 - d_{\mathbb{J}} \cdot u_1 \cdot u_2 \cdot v_1 \cdot v_2}$$

We use an optimized implementation found by Daira Hopwood making use of an observation by Bernstein and Lange in [BL2017, last paragraph of section 4.5.2]:

$$\begin{aligned}
(u_1 + v_1) \;\mathsf{x}\; (v_2 - a_{\mathbb{J}} \cdot u_2) &= (T) \\
(u_1) \;\mathsf{x}\; (v_2) &= (A) \\
(v_1) \;\mathsf{x}\; (u_2) &= (B) \\
(d_{\mathbb{J}} \cdot A) \;\mathsf{x}\; (B) &= (C) \\
(1 + C) \;\mathsf{x}\; (u_3) &= (A + B) \\
(1 - C) \;\mathsf{x}\; (v_3) &= (T - A + a_{\mathbb{J}} \cdot B)
\end{aligned}$$

The correctness of this implementation can be seen by expanding $T - A + a_{\mathbb{J}} \cdot B$:

$$\begin{aligned} T - A + a_{\mathbb{J}} \cdot B &= (u_1 + v_1) \cdot (v_2 - a_{\mathbb{J}} \cdot u_2) - u_1 \cdot v_2 + a_{\mathbb{J}} \cdot v_1 \cdot u_2 \\ &= v_1 \cdot v_2 - a_{\mathbb{J}} \cdot u_1 \cdot u_2 + u_1 \cdot v_2 - a_{\mathbb{J}} \cdot v_1 \cdot u_2 - u_1 \cdot v_2 + a_{\mathbb{J}} \cdot v_1 \cdot u_2 \\ &= v_1 \cdot v_2 - a_{\mathbb{J}} \cdot u_1 \cdot u_2 \end{aligned}$$

The above addition formulae are "unified", that is, they can also be used for doubling. Affine ctEdwards doubling $[2]\,(u, v) = (u_3, v_3)$ can also be implemented slightly more efficiently as:

$$\begin{aligned} &\big(u + v\big) \;\mathsf{x}\; \big(v - a_{\mathbb{J}} \cdot u\big) \;=\; \big(T\big) \\ &\big(u\big) \;\mathsf{x}\; \big(v\big) \;=\; \big(A\big) \\ &\big(d_{\mathbb{J}} \cdot A\big) \;\mathsf{x}\; \big(A\big) \;=\; \big(C\big) \\ &\big(1 + C\big) \;\mathsf{x}\; \big(u_3\big) \;=\; \big(2 \cdot A\big) \\ &\big(1 - C\big) \;\mathsf{x}\; \big(v_3\big) \;=\; \big(T + (a_{\mathbb{J}} - 1) \cdot A\big) \end{aligned}$$

This implementation is obtained by specializing the addition formulae to $(u, v) = (u_1, v_1) = (u_2, v_2)$ and observing that $u \cdot v = A = B$.

B.3.6 Affine ctEdwards nonsmall-order check

In order to avoid small-subgroup attacks on some protocols, it is useful to be able to check that a point is not of small order. Since this normally needs to be done in combination with a check that the coordinates are on the curve (Section B.3.1), we combine the two operations.

TODO: this was originally defined for Jubjub (with $h = 8$); check generalization.

ctEdwards curves have a large prime-order subgroup with a cofactor $h = 2^c$ where $c \in \{2, 3\}$. To check for a point P of order h or less, we can double c times (as in Section B.3.5) and checks that the resulting u-coordinate is not 0 (as in Section B.1.4).

On a ctEdwards curve, only the zero point $\mathcal{O}_{\mathbb{J}}$, and the unique point of order 2 at $(0, -1)$ have zero u-coordinate. The point of order 2 cannot occur as the result of c doublings. So this u-coordinate check rejects only $\mathcal{O}_{\mathbb{J}}$.

The total cost, including the curve check, is $5 \cdot c + 5$ constraints.

Note: This *does not* ensure that the point is in the prime-order subgroup.

Non-normative notes:

- It would have been sufficient to do $c - 1$ doublings, because the check that the u-coordinate is nonzero would reject both $\mathcal{O}_{\mathbb{J}}$ and the point of order 2.
- It is possible to reduce the cost to 8 constraints for the case of $c = 3$, by eliminating the redundant constraint in the curve point check (as mentioned in Section B.3.1); merging the first doubling with the curve point check; and then optimizing the second doubling based on the fact that we only need to check whether the resulting u-coordinate is zero.

B.3.7 Fixed-base affine ctEdwards scalar multiplication

If the base point B is fixed for a given scalar multiplication $[k]\,B$, we can fully precompute window tables for each window position.

TODO: generalize to non-Jubjub curves.

It is most efficient to use 3-bit fixed windows. Since the length of $r_{\mathbb{J}}$ is 252 bits, we need 84 windows.

Express k in base 8, i.e. $k = \sum_{i=0}^{83} k_i \cdot 8^i$.

Then $[k]\,B = \sum_{i=0}^{83} w_{(B,\,i,\,k_i)}$, where $w_{(B,\,i,\,k_i)} = [k_i \cdot 8^i]\,B$.

We precompute all of $w_{(B,\,i,\,s)}$ for $i \in \{0\,..\,83\}, s \in \{0\,..\,7\}$.

To look up a given window entry $w_{(B,\,i,\,s)} = (u_s, v_s)$, where $s = 4 \cdot s_2 + 2 \cdot s_1 + s_0$, we use:

$$(s_1)\ \mathsf{x}\ (s_2)\ =\ (s_{\&})$$

$$\begin{aligned}(s_0)\ \mathsf{x}\ \big(&- u_0 \cdot s_{\&} + u_0 \cdot s_2 + u_0 \cdot s_1 - u_0 + u_2 \cdot s_{\&} - u_2 \cdot s_1 + u_4 \cdot s_{\&} - u_4 \cdot s_2 - u_6 \cdot s_{\&} \\ &+ u_1 \cdot s_{\&} - u_1 \cdot s_2 - u_1 \cdot s_1 + u_1 - u_3 \cdot s_{\&} + u_3 \cdot s_1 - u_5 \cdot s_{\&} + u_5 \cdot s_2 + u_7 \cdot s_{\&}\big) = \\ \big(&u_s - u_0 \cdot s_{\&} + u_0 \cdot s_2 + u_0 \cdot s_1 - u_0 + u_2 \cdot s_{\&} - u_2 \cdot s_1 + u_4 \cdot s_{\&} - u_4 \cdot s_2 - u_6 \cdot s_{\&}\big)\end{aligned}$$

$$\begin{aligned}(s_0)\ \mathsf{x}\ \big(&- v_0 \cdot s_{\&} + v_0 \cdot s_2 + v_0 \cdot s_1 - v_0 + v_2 \cdot s_{\&} - v_2 \cdot s_1 + v_4 \cdot s_{\&} - v_4 \cdot s_2 - v_6 \cdot s_{\&} \\ &+ v_1 \cdot s_{\&} - v_1 \cdot s_2 - v_1 \cdot s_1 + v_1 - v_3 \cdot s_{\&} + v_3 \cdot s_1 - v_5 \cdot s_{\&} + v_5 \cdot s_2 + v_7 \cdot s_{\&}\big) = \\ \big(&v_s - v_0 \cdot s_{\&} + v_0 \cdot s_2 + v_0 \cdot s_1 - v_0 + v_2 \cdot s_{\&} - v_2 \cdot s_1 + v_4 \cdot s_{\&} - v_4 \cdot s_2 - v_6 \cdot s_{\&}\big)\end{aligned}$$

For a full-length (252-bit) scalar this costs 3 constraints for each of 84 window lookups, plus 6 constraints for each of 83 ctEdwards additions (as in Section B.3.5), for a total of 750 constraints. This does not include the cost of boolean-constraining the scalar.

Non-normative note: It would be more efficient to use arithmetic on the Montgomery curve, as in Section B.3.9. TODO: describe `https://github.com/zcash/zcash/issues/3924`.

B.3.8 Variable-base affine ctEdwards scalar multiplication

When the base point B is not fixed, the method in the preceding section cannot be used. Instead we use a naïve double-and-add method.

Given $k = \sum_{i=0}^{250} k_i \cdot 2^i$, we calculate $R = [k]\,B$ using:

// $\mathsf{Base}_i = [2^i]\,B$
let $\mathsf{Base}_0 = B$
let $\mathsf{Acc}_0^u = k_0\ ?\ \mathsf{Base}_0^u : 0$
let $\mathsf{Acc}_0^v = k_0\ ?\ \mathsf{Base}_0^v : 1$

for i from 1 up to 250:
  let $\mathsf{Base}_i = [2]\,\mathsf{Base}_{i-1}$
  // select Base_i or $\mathcal{O}_{\mathbb{J}}$ depending on the bit k_i
  let $\mathsf{Addend}_i^u = k_i\ ?\ \mathsf{Base}_i^u : 0$
  let $\mathsf{Addend}_i^v = k_i\ ?\ \mathsf{Base}_i^v : 1$
  let $\mathsf{Acc}_i = \mathsf{Acc}_{i-1} + \mathsf{Addend}_i$
let $R = \mathsf{Acc}_{250}$.

This costs 5 constraints for each of 250 ctEdwards doublings, 6 constraints for each of 250 ctEdwards additions, and 2 constraints for each of 251 point selections, for a total of 3252 constraints.

Non-normative note: It would be more efficient to use 2-bit fixed windows, and/or to use arithmetic on the Montgomery curve in a similar way to Section B.3.9.

B.3.9 Bowe–Hopwood Pedersen hash

The specification of Bowe–Hopwood Pedersen hashes is given in Section 6.1.2. It is based on the scheme from [CvHP1991, section 5.2] –for which a tighter security reduction to the Discrete Logarithm Problem was given in [BGG1995]– but tailored to allow several optimizations in the circuit implementation.

Since hash functions are often the bottleneck in applications (such as Zcash Sapling), this motivates considerable attention to optimizing the circuit implementation of this primitive, even at the cost of complexity.

First, we use a windowed scalar multiplication algorithm with signed digits. Each 3-bit message chunk corresponds to a window; the chunk is encoded as an integer from the set $\mathsf{Digits} = \{-4\,..\,4\} \setminus \{0\}$. This allows a more efficient lookup of the window entry for each chunk than if the set $\{1\,..\,8\}$ had been used, because a point can be conditionally negated using only a single constraint.

Next, we optimize the cost of point addition by allowing as many additions as possible to be performed on the Montgomery curve. An incomplete Montgomery addition costs 3 constraints, in comparison with a ctEdwards addition which costs 6 constraints.

However, we cannot do all additions on the Montgomery curve because the Montgomery addition is incomplete. In order to be able to prove that exceptional cases do not occur, we need to ensure that the distinct-x criterion from Section B.3.4 is met. This requires splitting the input into segments (each using an independent generator), calculating an intermediate result for each segment, and then converting to the ctEdwards curve and summing the intermediate results using ctEdwards addition.

Abstracting away the changes of curve, this calculation can be written as:

$$\mathsf{PedersenHashToPoint}(D, M) = \sum_{j=1}^{N} [\langle M_j \rangle]\, \mathcal{I}_j^D$$

where $\langle \bullet \rangle$ and $\mathcal{I}_j^D$ are defined as in Section 6.1.2.

We have to prove that:

- the Montgomery-to-ctEdwards conversions can be implemented without exceptional cases;
- the distinct-x criterion is met for all Montgomery additions within a segment.

The proof in Section 6.1.2 showed that all indices of addition inputs are in the range $\left\{-\frac{r_{\mathbb{J}}-1}{2}\,..\,\frac{r_{\mathbb{J}}-1}{2}\right\} \setminus \{0\}$.

Because the $\mathcal{I}_j^D$ (which are outputs of $\mathsf{GroupHash}^{\mathbb{J}^{(r)*}}$) are all of prime order, and $\langle M_j \rangle \neq 0 \pmod{r_{\mathbb{J}}}$, it is guaranteed that all of the terms $[\langle M_j \rangle]\, \mathcal{I}_j^D$ to be converted to ctEdwards form are of prime order. From the proof in Section B.3.3, we can infer that the conversions will not encounter exceptional cases.

We also need to show that the indices of addition inputs are all distinct disregarding sign.

Theorem B.3.4. *For all disjoint nonempty subsets S and S' of $\{1\,..\,c\}$, all $m \in \mathbb{B}^{[3][c]}$, and all $\Theta \in \{-1, 1\}$:*

$$\sum_{j \in S} \mathsf{enc}(m_j) \cdot 2^{4\cdot(j-1)} \neq \Theta \cdot \sum_{j' \in S'} \mathsf{enc}(m_{j'}) \cdot 2^{4\cdot(j'-1)}.$$

Proof. Suppose for a contradiction that S, S', m, Θ is a counterexample. Taking the multiplication by Θ on the right hand side inside the summation, we have:

$$\sum_{j \in S} \mathsf{enc}(m_j) \cdot 2^{4\cdot(j-1)} = \sum_{j' \in S'} \Theta \cdot \mathsf{enc}(m_{j'}) \cdot 2^{4\cdot(j'-1)}.$$

Define $\mathsf{enc}' : \{-1, 1\} \times \mathbb{B}^{[3]} \to \{0\,..\,8\} \setminus \{4\}$ as $\mathsf{enc}'_\theta(m_i) := 4 + \theta \cdot \mathsf{enc}(m_i)$.

Let $\Delta = 4 \cdot \sum_{i=1}^{c} 2^{4\cdot(i-1)}$ as in the proof in Section 6.1.2. By adding Δ to both sides, we get

$$\sum_{j \in S} \mathsf{enc}'_1(m_j) \cdot 2^{4\cdot(j-1)} + \sum_{j \in \{1\,..\,c\} \setminus S} 4 \cdot 2^{4\cdot(j-1)} = \sum_{j' \in S'} \mathsf{enc}'_\Theta(m_{j'}) \cdot 2^{4\cdot(j'-1)} + \sum_{j' \in \{1\,..\,c\} \setminus S'} 4 \cdot 2^{4\cdot(j'-1)}$$

where all of the $\mathsf{enc}'_1(m_j)$ and $\mathsf{enc}'_\Theta(m_{j'})$ are in $\{0\,..\,8\} \setminus \{4\}$.

Each term on the left and on the right affects the single hex digit indexed by j and j' respectively. Since S and S' are disjoint subsets of $\{1\,..\,c\}$ and S is nonempty, $S \cap (\{1\,..\,c\} \setminus S')$ is nonempty. Therefore the left hand side has at least one hex digit not equal to 4 such that the corresponding right hand side digit is 4; contradiction. □

This implies that the terms in the Montgomery addition –as well as any intermediate results formed from adding a distinct subset of terms– have distinct indices disregarding sign, hence distinct x-coordinates by the proof in Section B.3.4. (We make no assumption about the order of additions.)

We now describe the subcircuit used to process each chunk, which contributes most of the constraint cost of the hash. This subcircuit is used to perform a lookup of a Montgomery point in a 2-bit window table, conditionally negate the result, and add it to an accumulator holding another Montgomery point.

Suppose that the bits of the chunk, $[s_0, s_1, s_2]$, are already boolean-constrained.

We aim to compute $C = A + [(1 - 2 \cdot s_2) \cdot (1 + s_0 + 2 \cdot s_1)]\,P$ for some fixed base point P and accumulated sum A.

We first compute $s_\& = s_0 \wedge s_1$:

$$(s_0)\ \mathsf{X}\ (s_1)\ =\ (s_\&)$$

Let $(x_k, y_k) = [k]\,P$ for $k \in \{1\,..\,4\}$. Define each coordinate of $(x_S, y_R) = [1 + s_0 + 2 \cdot s_1]\,P$ as a linear combination of s_0, s_1, and $s_\&$:

$$\text{let } x_S = x_1 + (x_2 - x_1) \cdot s_0 + (x_3 - x_1) \cdot s_1 + (x_4 + x_1 - x_2 - x_3) \cdot s_\&$$
$$\text{let } y_R = y_1 + (y_2 - y_1) \cdot s_0 + (y_3 - y_1) \cdot s_1 + (y_4 + y_1 - y_2 - y_3) \cdot s_\&$$

We implement the conditional negation as $(2 \cdot y_R)\ \mathsf{X}\ (s_2)\ =\ (y_R - y_S)$. After substitution of y_R this becomes:

$$\begin{aligned}&\big(2 \cdot (y_1 + (y_2 - y_1) \cdot s_0 + (y_3 - y_1) \cdot s_1 + (y_4 + y_1 - y_2 - y_3) \cdot s_\&)\big)\ \mathsf{X}\ (s_2) = \\ &\quad \big(y_1 + (y_2 - y_1) \cdot s_0 + (y_3 - y_1) \cdot s_1 + (y_4 + y_1 - y_2 - y_3) \cdot s_\& - y_S\big)\end{aligned}$$

Then we substitute x_S into the Montgomery addition constraints from Section B.3.4, as follows:

$$\big(x_1 + (x_2 - x_1) \cdot s_0 + (x_3 - x_1) \cdot s_1 + (x_4 + x_1 - x_2 - x_3) \cdot s_\& - x_A\big)\ \mathsf{X}\ (\lambda)\ =\ (y_S - y_A)$$
$$(B_\mathbb{M} \cdot \lambda)\ \mathsf{X}\ (\lambda)\ =\ \big(A_\mathbb{M} + x_A + x_1 + (x_2 - x_1) \cdot s_0 + (x_3 - x_1) \cdot s_1 + (x_4 + x_1 - x_2 - x_3) \cdot s_\& + x_C\big)$$
$$(x_A - x_C)\ \mathsf{X}\ (\lambda)\ =\ (y_C + y_A)$$

For the first addition in each segment, both sides are looked up and substituted into the Montgomery addition, so the first lookup takes only 2 constraints.

If part of the input is fixed, then a precomputation may be possible for some of the windows.

The cost of a Pedersen hash over ℓ bits (where ℓ includes the fixed bits) is as follows. The number of chunks is $c = \mathsf{ceiling}\left(\frac{\ell}{3}\right)$ and the number of segments is $n = \mathsf{ceiling}\left(\frac{\ell}{3 \cdot 63}\right)$.

The cost is then:

- $2 \cdot c$ constraints for the lookups;
- $3 \cdot (c - n)$ constraints for incomplete additions on the Montgomery curve;
- $2 \cdot n$ constraints for Montgomery-to-ctEdwards conversions;
- $6 \cdot (n - 1)$ constraints for ctEdwards additions;

for a total of $5 \cdot c + 5 \cdot n - 6$ constraints. This does not include the cost of boolean-constraining inputs.

C Batching Optimizations

C.1 RedDSA batch verification

The reference verification algorithm for RedDSA signatures is defined in Section 6.3.

Let the RedDSA parameters $\mathbb{G}$ (defining a subgroup $\mathbb{G}^{(r)}$ of order $r_{\mathbb{G}}$, a cofactor $h_{\mathbb{G}}$, a group operation $+$, an additive identity $\mathcal{O}_{\mathbb{G}}$, a bit-length $\ell_{\mathbb{G}}$, a representation function $\mathsf{repr}_{\mathbb{G}}$, and an abstraction function $\mathsf{abst}_{\mathbb{G}}$); $\mathcal{P}_{\mathbb{G}} : \mathbb{G}$; $\ell_{\mathsf{H}} : \mathbb{N}$; $\mathsf{H} : \mathbb{B}^{\mathbb{Y}[\mathbb{N}]} \to \mathbb{B}^{\mathbb{Y}[\ell_{\mathsf{H}}/8]}$; and the derived hash function $\mathsf{H}^{\circledast} : \mathbb{B}^{\mathbb{Y}[\mathbb{N}]} \to \mathbb{F}_{r_{\mathbb{G}}}$ be as defined in that section.

Implementations can alternatively use the optimized procedure described in this section to perform faster verification of a batch of signatures, i.e. to determine whether all signatures in a batch are valid. Its input is a sequence of N *signature batch entries*, each of which is a (public key, message, signature) triple.

Let $\mathsf{LEOS2BSP}$, $\mathsf{LEOS2IP}$, and $\mathsf{LEBS2OSP}$ be as defined in Section 5.

Define $\mathsf{RedDSA.BatchEntry} := \mathsf{RedDSA.Public} \times \mathsf{RedDSA.Message} \times \mathsf{RedDSA.Signature}$.

Define $\mathsf{RedDSA.BatchValidate} : (\mathsf{entry}_{0\,..\,N-1} : \mathsf{RedDSA.BatchEntry}^{[N]}) \to \mathbb{B}$ as:

For each $j \in \{0\,..\,N-1\}$:

Let $(\mathsf{vk}_j, M_j, \sigma_j) = \mathsf{entry}_j$.

Let $\underline{R_j}$ be the first $\mathsf{ceiling}(\ell_{\mathbb{G}}/8)$ bytes of σ_j, and let $\underline{S_j}$ be the remaining $\mathsf{ceiling}(\mathsf{bitlength}(r_{\mathbb{G}})/8)$ bytes.

Let $R_j = \mathsf{abst}_{\mathbb{G}}\big(\mathsf{LEOS2BSP}_{\ell_{\mathbb{G}}}(\underline{R_j})\big)$, and let $S_j = \mathsf{LEOS2IP}_{8\cdot\mathsf{length}(\underline{S_j})}(\underline{S_j})$.

Let $\underline{\mathsf{vk}_j} = \mathsf{LEBS2OSP}_{\ell_{\mathbb{G}}}(\mathsf{repr}_{\mathbb{G}}(\mathsf{vk}_j))$.

Let $c_j = \mathsf{H}^{\circledast}(\underline{R_j} \,||\, \underline{\mathsf{vk}_j} \,||\, M_j)$.

Choose random $z_j : \mathbb{F}^*_{r_{\mathbb{G}}} \xleftarrow{\mathrm{R}} \{1\,..\,2^{128}-1\}$.

Return 1 if

- for all $j \in \{0\,..\,N-1\}$, $R_j \neq \perp$ and $S_j < r_{\mathbb{G}}$; and
- $[h_{\mathbb{G}}]\left(\left[\sum_{j=0}^{N-1}(z_j \cdot S_j) \pmod{r_{\mathbb{G}}}\right]\mathcal{P}_{\mathbb{G}} + \sum_{j=0}^{N-1}\left([z_j]\,R_j + [z_j \cdot c_j \pmod{r_{\mathbb{G}}}]\,\mathsf{vk}_j\right)\right) = \mathcal{O}_{\mathbb{G}}$,

otherwise 0.

The z_j values must be chosen independently of the signature batch entries.

The performance benefit of this approach arises partly from replacing the per-signature scalar multiplication of the base $\mathcal{P}_{\mathbb{G}}$ with one such multiplication per batch, and partly from using an efficient algorithm for multiscalar multiplication such as Pippinger's method [Bernstein2001] or the Bos–Coster method [deRooij1995], as explained in [BDLSY2012, section 5].

It is straightforward to adapt the above procedure to handle multiple bases; there is one $\left[\sum_j (z_j \cdot S_j) \pmod{r_{\mathbb{G}}}\right]\mathcal{P}$ term for each base $\mathcal{P}$. The benefit of this relative to using separate batches is that the multiscalar multiplication can be extended across a larger batch.

C.2 Groth16 batch verification

TODO: reference the single verification algorithm.

Let $q_{\mathbb{S}}$, $r_{\mathbb{S}}$, $\mathbb{S}^{(r)}_{1,2,T}$, $\mathbb{S}^{(r)*}_{1,2,T}$, $\mathcal{P}_{\mathbb{S}_{1,2,T}}$, $\mathbf{1}_{\mathbb{S}}$, and $\hat{e}_{\mathbb{S}}$ be as defined in Section 3.9.

Define $\mathsf{MillerLoop}_{\mathbb{S}} : \mathbb{S}^{(r)}_1 \times \mathbb{S}^{(r)}_2 \to \mathbb{S}^{(r)}_T$ and $\mathsf{FinalExp}_{\mathbb{S}} : \mathbb{S}^{(r)}_T \to \mathbb{S}^{(r)}_T$ to be the Miller loop and final exponentiation respectively of the $\hat{e}_{\mathbb{S}}$ pairing computation, so that:

$\hat{e}_{\mathbb{S}}(P, Q) = \mathsf{FinalExp}_{\mathbb{S}}(\mathsf{MillerLoop}_{\mathbb{S}}(P, Q))$

where $\mathsf{FinalExp}_{\mathbb{S}}(R) = R^t$ for some fixed t.

Define $\mathsf{Groth16}_{\mathbb{S}}.\mathsf{Proof} := \mathbb{S}_1^{(r)*} \times \mathbb{S}_2^{(r)*} \times \mathbb{S}_1^{(r)*}$.

A $\mathsf{Groth16}_{\mathbb{S}}$ proof consists of a tuple $(\pi_A, \pi_B, \pi_C) : \mathsf{Groth16}_{\mathbb{S}}.\mathsf{Proof}$.

Verification of a single $\mathsf{Groth16}_{\mathbb{S}}$ proof against an instance encoded as $a_{0\,..\,\ell} : \mathbb{F}^{[\ell+1]}$ requires checking the equation

$$\hat{e}_{\mathbb{S}}(\pi_A, \pi_B) = \hat{e}_{\mathbb{S}}(\pi_C, \Delta) \cdot \hat{e}_{\mathbb{S}}\Big(\sum\nolimits_{i=0}^{\ell} [a_i]\, \Psi_i, \Gamma\Big) \cdot Y$$

where $\Delta = [\delta]\, \mathcal{P}_{\mathbb{S}_2}, \Gamma = [\gamma]\, \mathcal{P}_{\mathbb{S}_2}$, $Y = [\alpha \cdot \beta]\, \mathcal{P}_{\mathbb{S}_T}$, and $\Psi_i = \Big[\frac{\beta \cdot u_i(x) + \alpha \cdot v_i(x) + w_i(x)}{\gamma}\Big] \mathcal{P}_{\mathbb{S}_1}$ for $i \in \{0\,..\,\ell\}$ are elements of the verification key, as described (with slightly different notation) in [Groth2016, section 3.2].

This can be written as:

$$\hat{e}_{\mathbb{S}}(\pi_A, -\pi_B) \cdot \hat{e}_{\mathbb{S}}(\pi_C, \Delta) \cdot \hat{e}_{\mathbb{S}}\Big(\sum\nolimits_{i=0}^{\ell} [a_i]\, \Psi_i, \Gamma\Big) \cdot Y = \mathbf{1}_{\mathbb{S}}.$$

Raising to the power of random $z \neq 0$ gives:

$$\hat{e}_{\mathbb{S}}([z]\, \pi_A, -\pi_B) \cdot \hat{e}_{\mathbb{S}}([z]\, \pi_C, \Delta) \cdot \hat{e}_{\mathbb{S}}\Big(\sum\nolimits_{i=0}^{\ell} [z \cdot a_i]\, \Psi_i, \Gamma\Big) \cdot Y^z = \mathbf{1}_{\mathbb{S}}.$$

This justifies the following optimized procedure for performing faster verification of a batch of $\mathsf{Groth16}_{\mathbb{S}}$ proofs. Implementations can use this procedure to determine whether all proofs in a batch are valid.

Define $\mathsf{Groth16}_{\mathbb{S}}.\mathsf{BatchEntry} := \mathsf{Groth16}_{\mathbb{S}}.\mathsf{Proof} \times \mathsf{Groth16}_{\mathbb{S}}.\mathsf{PrimaryInput}$.

Define $\mathsf{Groth16}_{\mathbb{S}}.\mathsf{BatchVerify} : (\mathsf{entry}_{0\,..\,N-1} : \mathsf{Groth16}_{\mathbb{S}}.\mathsf{BatchEntry}^{[N]}) \to \mathbb{B}$ as:

For each $j \in \{0\,..\,N-1\}$:

Let $((\pi_{j,A},\ \pi_{j,B},\ \pi_{j,C}),\ a_{j,\,0\,..\,\ell}) = \mathsf{entry}_j$.

Choose random $z_j : \mathbb{F}^*_{\xleftarrow{\mathrm{R}}} \{1\,..\,2^{128} - 1\}$.

Let $\mathsf{Accum}_{AB} = \prod_{j=0}^{N-1} \mathsf{MillerLoop}_{\mathbb{S}}([z_j]\, \pi_{j,A}, -\pi_{j,B})$.

Let $\mathsf{Accum}_{\Delta} = \sum_{j=0}^{N-1} [z_j]\, \pi_{j,C}$.

Let $\mathsf{Accum}_{\Gamma,i} = \sum_{j=0}^{N-1} (z_j \cdot a_{j,i}) \pmod{r_{\mathbb{S}}}$ for $i \in \{0\,..\,\ell\}$.

Let $\mathsf{Accum}_{Y} = \sum_{j=0}^{N-1} z_j \pmod{r_{\mathbb{S}}}$.

Return 1 if

$$\mathsf{FinalExp}_{\mathbb{S}}\left(\mathsf{Accum}_{AB} \cdot \mathsf{MillerLoop}_{\mathbb{S}}\big(\mathsf{Accum}_{\Delta}, \Delta\big) \cdot \mathsf{MillerLoop}_{\mathbb{S}}\Big(\sum\nolimits_{i=0}^{\ell} [\mathsf{Accum}_{\Gamma,i}]\, \Psi_i, \Gamma\Big)\right) \cdot Y^{\mathsf{Accum}_Y} = \mathbf{1}_{\mathbb{S}},$$

otherwise 0.

The z_j values must be chosen independently of the proof batch entries.

The performance benefit of this approach arises from computing two of the three Miller loops, and the final exponentation, per batch instead of per proof. For the multiplications by z_j, an efficient algorithm for multiscalar multiplication such as Pippinger's method [Bernstein2001] or the Bos–Coster method [deRooij1995] may be used.

Note: It is straightforward to adapt the above procedure to handle multiple verification keys; the accumulator variables Accum_{Δ}, $\mathsf{Accum}_{\Gamma,i}$, and Accum_{Y} are duplicated, with one term in the verification equation for each variable, while Accum_{AB} is shared.

Neglecting multiplications in $\mathbb{S}_T^{(r)}$ and $\mathbb{F}$, and other trivial operations, the cost of batched verification is therefore

- for each proof: the cost of decoding the proof representation to the form $\mathsf{Groth16}_{\mathbb{S}}.\mathsf{Proof}$, which requires three point decompressions and three subgroup checks (two for $\mathbb{S}_1^{(r)*}$ and one for $\mathbb{S}_2^{(r)*}$);

- for each successfully decoded proof: a Miller loop; and a 128-bit scalar multiplication by z_j in $\mathbb{S}_1^{(r)}$;
- for each verification key: two Miller loops; an exponentiation in $\mathbb{S}_T^{(r)}$; a multiscalar multiplication in $\mathbb{S}_1^{(r)}$ with N 128-bit scalars to compute Accum_Δ; and a multiscalar multiplication in $\mathbb{S}_1^{(r)}$ with $\ell + 1$ 255-bit scalars to compute $\sum_{i=0}^{\ell} [\mathsf{Accum}_{\Gamma,i}]\, \Psi_i$;
- one final exponentiation.