

Hashing to Curves: From Theory to RFC 9380 Specification

Armando Faz Hernandez
armfazh@cloudflare.com

Joint work with: Sam Scott, Nick Sullivan, Riad S. Wahby, Christopher A. Wood

NIST Crypto Reading Club
February 18th, 2026

Outline

- I: Hash to Curve Function
- II: Design Considerations
- III: RFC 9380 Specification
- IV: Protocols

Part I

Hash to Curve Function

Hash Function

Elliptic Curves

Hash to Curve

Insecure Methods

Real-World Attack

Part I – Hash to Curve Function

Hash Function

Elliptic Curves

Hash to Curve

Insecure Methods

Real-World Attack

Cryptographic Hash Function

A cryptographic hash function

$$H: \{0, 1\}^* \rightarrow \{0, 1\}^l$$

has the following properties:

Preimage resistance

Given y , find x such that $H(x) = y$.

2nd-preimage resistance

Given x , find x' such that $H(x) = H(x')$ and $x \neq x'$.

Collision resistance

Find pairs (x, x') such that $H(x) = H(x')$ and $x \neq x'$.

Since H has infinite domain, collisions necessarily exist, but it must be difficult to find them.

Standard Hash Functions

Merkle-Damgård construction.

- SHA-2 family.

Name	Block Size	Output Size
SHA-256	512	256
SHA-384	1024	384
SHA-512	1024	512

Sponge-based construction.

- SHA-3 family.

Name	Block Size	Output Size
SHA3-256	1088	256
SHA3-384	832	384
SHA3-512	576	512

- SHA-3 (FIPS 202) defines extendable-output functions.

Part I – Hash to Curve Function

Hash Function

Elliptic Curves

Hash to Curve

Insecure Methods

Real-World Attack

Hashing to
Curves:
From Theory to
RFC 9380
Specification

Armando Faz

Hash Function

Elliptic Curves

Hash to Curve

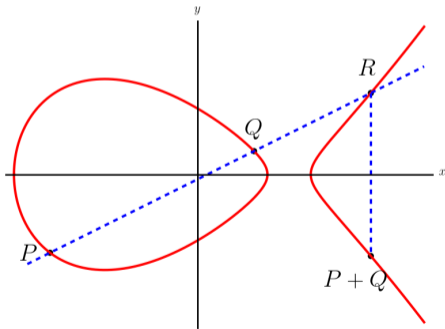
Insecure Methods

Real-World Attack

Elliptic Curves

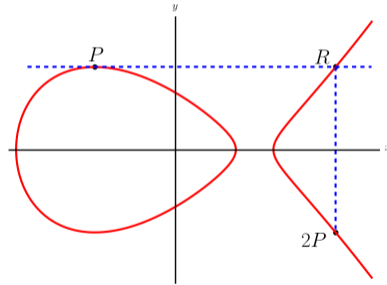
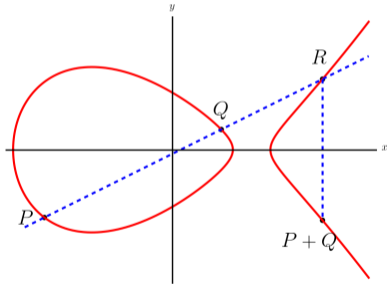
An elliptic curve is defined over a finite field \mathbb{F} by the Weierstrass equation:

$$E/\mathbb{F}: y^2 = x^3 + ax + b$$



Point Addition

Given two points P and Q on the curve, the *chord-and-tangent* rule produces a third point R also on the curve.



The Group of Points on an Elliptic Curve

The points on an elliptic curve form an additive group with O as the identity.

$$E(\mathbb{F}) = \{(x, y) : y^2 - x^3 - ax - b = 0\} \cup \{O\},$$

The group denoted as $\mathbb{G} = (E(\mathbb{F}), +, O)$.

Group Operations

- Point Addition: Given P and Q calculate $P + Q$.
- Scalar Multiplication: Given P and an integer k calculate

$$kP = P + P + \dots + P, \quad \text{where } P \text{ appears } k \text{ times.}$$

The Group of Points on an Elliptic Curve

Group Order

The number of points on the group.

$$|E(\mathbb{F})| = \begin{cases} q, & \text{large prime number.} \\ hq, & \text{a multiple of a large prime, } h \text{ is cofactor.} \end{cases}$$

Group Generator

Let $\langle P \rangle = \{0P, 1P, 2P, \dots\}$ be the points generated by P .
If $\mathbb{G} = \langle G \rangle$, then G is a generator of the group.

Pairing

A bilinear map

$$e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mu_q$$

where $\mathbb{G}_i \subsetneq E(\mathbb{F})$ are subgroups of order q , and μ_q are the q th roots of unity.

Short Signatures (BLS-2001)

Key Generation

Private key is $k \leftarrow_{\$} \{1, \dots, n\}$, and public key is $Q = kG$.

Signing m

Assume $H: \{0, 1\}^* \rightarrow \mathbb{G}$ modeled as a random oracle.

$$P = H(m)$$

$$S = kP$$

Verify

Let $P = H(m)$, accept if $e(S, G) = e(P, Q)$; otherwise, reject.

Correctness

$$e(S, G) = e(P, Q)$$

$$e(kP, G) = e(P, kG)$$

$$e(P, G)^k = e(P, G)^k$$

Part I – Hash to Curve Function

Hash Function

Elliptic Curves

Hash to Curve

Insecure Methods

Real-World Attack

Hashing to
Curves:
From Theory to
RFC 9380
Specification

Armando Faz

Hash Function

Elliptic Curves

Hash to Curve

Insecure Methods

Real-World Attack

Hash to Curve

Problem

Given an arbitrary-large bit string $\{0, 1\}^*$ and an elliptic curve

$$E/\mathbb{F}: y^2 = x^3 + ax + b$$

Goal

Define a cryptographic hash function H that outputs a point on the curve.

$$H: \{0, 1\}^* \rightarrow \mathbb{G}$$

Properties:

- The same as for standard hash functions.
- Deterministic algorithm.
- Countermeasures against side-channel attacks.
- Formal specification, efficient implementations, test vectors, libraries.

Basic Template

Input $m \in \{0, 1\}^*$

Procedure

- $h = H(m)$, where H is a standard hash function, e.g. SHA-256.
- MapToField takes h and returns $u \in \mathbb{F}$.
- MapToCurve takes u and returns a point $(x, y) \in E(\mathbb{F})$.

Part I – Hash to Curve Function

Hash Function

Elliptic Curves

Hash to Curve

Insecure Methods

Real-World Attack

Hashing to
Curves:
From Theory to
RFC 9380
Specification

Armando Faz

Hash Function

Elliptic Curves

Hash to Curve

Insecure Methods

Real-World Attack

Insecure Method #1: Scalar Multiplication

Scalar Multiplication

- 1: Sample $k \leftarrow_{\$} \{0, \dots, q\}$
- 2: **return** kG

Known Discrete Logarithm. Some protocols, such as Identity-based Encryption (IBE), require that no relation between G and $H(\cdot)$ is known.

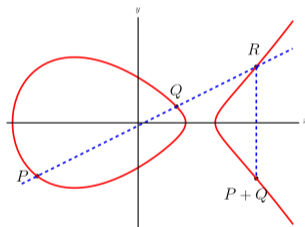
Insecure Method #2: Trial-and-Error

Define $g(x) = x^3 + ax + b$, so $y^2 = g(x)$.

Trial-and-Error

- 1: Sample $x \leftarrow_{\$} \mathbb{F}$
- 2: Calculate $g(x)$
- 3: **if** $g(x)$ is a quadratic residue (QR) **then**
 return $(x, \sqrt{g(x)})$
- 4: **end if**
- 5: Otherwise, fail.

Non-deterministic. Succeeds only when $g(x)$ is QR.



Insecure Method #3: Try-and-Increment

Try-and-Increment

- 1: Set $i = 0$
- 2: Sample $x \leftarrow_{\$} \mathbb{F}$
- 3: Calculate $g(x + i)$
- 4: **if** $g(x + i)$ is QR **then**
- 5: **return** $(x, \sqrt{g(x + i)})$
- 6: **else**
- 7: Increment i and go to Step 3.
- 8: **end if**

Non-constant Time. Succeeds after an unknown number of iterations.

Part I – Hash to Curve Function

Hash Function

Elliptic Curves

Hash to Curve

Insecure Methods

Real-World Attack

Hashing to
Curves:
From Theory to
RFC 9380
Specification

Armando Faz

Hash Function

Elliptic Curves

Hash to Curve

Insecure Methods

Real-World Attack

DragonBlood: Attack on WPA3 and EAP-pwd

WPA3 and EAP-pwd

Protocols used to authenticate users on Wi-Fi networks.

Dragonfly

A password-authenticated key exchange (PAKE) used in both protocols.

DragonBlood Attack

Vanhoef-Ronen (2019) showed an attack that exploits a vulnerable hash to curve function.

- Attacker access the Wi-Fi network without knowing the user's password.

<https://wpa3.mathyvanhoef.com/>

Insecure Method #4: Try-and-Increment-k-Times

Try-and-Increment-k-Times

```
1: for  $i = 1$  to  $k$  do  
2:   Set  $i = 0$   
3:   Sample  $x \leftarrow_{\$} \mathbb{F}$   
4:   Calculate  $y = g(x + i)$   
5:   if  $y$  is QR then  
6:     Save  $(x, y)$   
7:   else  
8:     Increment  $i$  and go to Step 3.  
9:   end if  
10: end for  
11: return  $(x, y)$ 
```

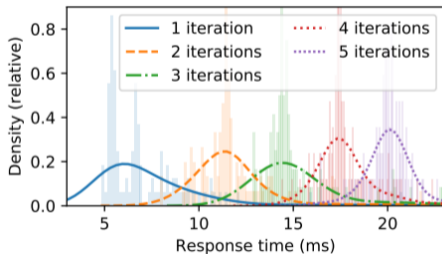
Inefficient Loops more times than needed.

DoS Attack Adversary can abuse the overhead of extra iterations.

DragonBlood: Timing Attack

[ia.cr/2019/383]

... we perform a timing attack against an iwd client using EAP-pwd with curve P-256. With EAP-pwd, the number of executed iterations are influenced by the client's username, the identity of the server, and by a token generated by the server.



(c) EAP-pwd client with curve P-256.

DragonBlood: Vulnerable Hash To Curve

Listing 1: Hash-to-curve method in Python-like pseudocode. If token is None the SAE variant is executed [46, §12.4.4.2.2], otherwise it executes the EAP-pwd variant [40].

```
1 def hash_to_curve(password, id1, id2, token=None):
2     found, counter, base = False, 0, password
3     label = "EAP-pwd" if token else "SAE"
4     k = 0 if token else 40
5     while counter < k or not found:
6         counter += 1
7         seed = Hash(token, id1, id2, base, counter)
8         value = KDF(seed, label + " Hunting and Pecking", p)
9         if value >= p: continue
10        if is_quadratic_residue(value^3 + a * value + b, p):
11            if not found:
12                x, save, found = value, seed, True
13                base = random()
14
15        y = sqrt(x^3 + a * x + b) mod p
16        P = (x, y) if LSB(save) == LSB(y) else (x, p - y)
17        return P
```

It returns immediately, once a point has been found.

The Need for Secure Hash to Curve Functions

- No concrete guidance about how to perform hash to curve securely.
- Papers on the topic are difficult to grasp.
- Many results come from algebraic-geometry area.
- Insecure algorithms are popular.

Part II

Design Considerations

Encoding vs Hash

Hash to Finite Field

Mapping to Curve

Clearing Cofactor

Hash to Curve

Part II – Design Considerations

Encoding vs Hash

Hash to Finite Field

Mapping to Curve

Clearing Cofactor

Hash to Curve

Encoding vs Hash

A function $\{0, 1\}^* \rightarrow \mathbb{G}$ is:

Encoding

- A nonuniform encoding from strings to points in \mathbb{G} .
- The distribution of its output is not uniformly random in \mathbb{G} .
- Only a fraction of the points in \mathbb{G} .
- Some points are more likely to be output than others.

Hash

- A uniform encoding from strings to points in \mathbb{G} .
- The distribution of its output is statistically close to uniform in \mathbb{G} .
- Suitable for applications requiring a random oracle returning points in \mathbb{G} .

Encoding

Goal: A function $\{0, 1\}^* \rightarrow \mathbb{G}$ that is an encoding.

Strategy:

Hash to Field

$$\{0, 1\}^* \rightarrow \mathbb{F}$$

Mapping to Curve

$$\mathbb{F} \rightarrow E(\mathbb{F})$$

Clear cofactor

$$E(\mathbb{F}) \rightarrow \mathbb{G}$$

Hash to Curve

Goal: A function $\{0, 1\}^* \rightarrow \mathbb{G}$ that is a hash.

Strategy: *it will be shown later...*

Part II – Design Considerations

Encoding vs Hash

Hash to Finite Field

Mapping to Curve

Clearing Cofactor

Hash to Curve

Sampling Prime Field Elements

For \mathbb{F}_p , in most cases p is close to a power of two, say $p \approx 2^b$.
Then one can sample elements from the set $\{0, \dots, 2^b - 1\}$.

This does not apply generally:

- Matching is not perfect, so there is a bias.
- The prime sizes do not match the output sizes of hash functions.
- One could use XOFs, but common hash functions are widely deployed.

Secure Sampling

Let k be the security level,
elements of \mathbb{F}_p are sampled from $\{0, \dots, 2^L - 1\}$,
where $L = \lceil \log(p) + k \rceil$.

For elliptic curves, $p \approx 2^{2k}$, so $L = 3k$ bits.
(Proposition 4, Brier et al. <https://eprint.iacr.org/2009/340>)

Standard hash functions do not output so many bits.

Expanders

- XMD Expander: based on a Merkle-Damgård hash function.
 - ▶ It follows an NMAC construction given by Coron et al.
https://doi.org/10.1007/11535218_26
 - ▶ e.g. SHA-256, SHA-512, ...

- XOF Expander: based on an extendable output function.
 - ▶ e.g. SHAKE128, SHAKE256, ...

Hash to Finite Field

Let k be the security level, for a message M (and DST)

hash_to_field produces n elements of \mathbb{F}_p

- 1: $(x_1, \dots, x_n) = \text{Expander}(M, \text{DST}, L \times n)$
- 2: $u_i = x_i \bmod p$ for $0 < i \leq n$
- 3: **return** (u_1, \dots, u_n)

Easy to extend to finite field extensions \mathbb{F}_{p^m} , just request m times more bytes.

Domain Separation

Protocols assume the random oracles are independent.

The implementation must avoid that queries are to the same oracle; but, hash functions can be reused.

Example:

$$h_0(M) = H(M \parallel 0)$$

$$h_1(M) = H(M \parallel 1)$$

are different oracles.

A **domain separation tag** (DST) can be used to instantiate different oracles.

Part II – Design Considerations

Encoding vs Hash

Hash to Finite Field

Mapping to Curve

Weierstrass Curves

Montgomery Curves

Twisted Edwards Curves

Clearing Cofactor

Hash to Curve

Hashing to
Curves:
From Theory to
RFC 9380
Specification

Armando Faz

Encoding vs Hash

Hash to Finite Field

Mapping to Curve

Weierstrass Curves

Montgomery Curves

Twisted Edwards
Curves

Clearing Cofactor

Hash to Curve

Deterministic Algorithms

A deterministic mapping takes an element in \mathbb{F} to generate a point on the elliptic curve.

Algorithms

- Skalba equations. Polynomials of large degree.
- Icart's method. Applies to $q \equiv 2 \pmod{3}$.
- Shallue-van de Woestijne, SSWU, Elligator2, Boneh-Franklin.

Shallue-van de Woestijne (SW) Mapping

Applies to almost all Weierstrass curves

$$y^2 = x^3 + ax + b$$

for some $Z \neq 0$ such that

- $g(Z) \neq 0$.
- $-(3Z^2 + 4A)/(4g(Z)) \neq 0$.
- $-(3Z^2 + 4A)/(4g(Z))$ is square in \mathbb{F} .
- At least one of $g(Z)$ and $g(-Z/2)$ is square in \mathbb{F} .

Cost: Takes at most 2 square roots.

Simplified SWU Mapping

Applies to Weierstrass curves with $ab \neq 0$

$$y^2 = x^3 + ax + b$$

for some $Z \neq 0$ such that

- Z is non-square in \mathbb{F} .
- $Z \neq -1 \in \mathbb{F}$.
- The polynomial $g(x) - Z$ is irreducible over \mathbb{F}
- $g(B/(ZA))$ is square in \mathbb{F} .

Cost: Takes 1 square root.

Simplified SWU Mapping

Mapping to E when $a = 0$ or $b = 0$.

- Find an isogeneous curve $E' : y^2 = x^3 + a'x + b'$ such that $a'b' \neq 0$.
- Use SSWU on E' .
- Apply the isogeny $E' \rightarrow E$.

Cost: Extra cost by performing the isogeny.

Elligator2 Mapping

Applies to Montgomery curves

$$E: y^2 = x^3 + Ax^2 + Bx$$

for some r a QNR with input u

- $v = -\frac{A}{1+ur^2}$
- $e = \text{Legendre} \left(\frac{g(v)}{p} \right)$
- $x = ev - (1 - e)A/2$
- $y = \sqrt{g(x)}$
- Return $(x, y) \in E$

Cost: 1 inversion, 1 square-root, 1 Legendre symbol.

This can be reduced to 1 exponentiation.

Twisted Edwards Curves

- Apply Elligator2 on the Montgomery curve associated.
- $Q = \text{Elligator2}(u) \in E_M$
- Use birational equivalence from $\phi: E_M \rightarrow E_T$
- $P = \phi(Q) \in E_T$

Part II – Design Considerations

Encoding vs Hash

Hash to Finite Field

Mapping to Curve

Clearing Cofactor

Hash to Curve

Hashing to
Curves:
From Theory to
RFC 9380
Specification

Armando Faz

Encoding vs Hash

Hash to Finite Field

Mapping to Curve

Clearing Cofactor

Hash to Curve

Issue in Curves with Cofactor

Recall that not all elliptic curves produce prime order groups.

The output of a mapping can be any point on the curve.

It is not guaranteed that the point be in a specific subgroup.

No action needed when $E(\mathbb{F})$ is a prime group (cofactor is 1).

However, some elliptic curves have cofactors.

Clearing Cofactor

If the order of the curve is hq ,
there exists G_1 and G_2 generators of order h and q .

Then, a point P produced by the mapping is

$$P = k_1G_1 + k_2G_2$$

The easiest way to get a point in the main subgroup is multiplying by h

$$\begin{aligned}hP &= hk_1G_1 + hk_2G_2 \\ &= \cancel{hk_1G_1} + hk_2G_2 \\ &= hk_2G_2 \in \mathbb{G}\end{aligned}$$

Clearing Cofactor

Two cases:

1. If P belongs to the subgroup of order h .

$$hP = O \in \mathbb{G}$$

2. If P already belongs to the subgroup of order q .

$$hP = hk_1G_2 \in \mathbb{G}$$

Cofactor multiplication is safe acting as a **permutation**.

Faster Cofactor Multiplication

Multiplying times the cofactor h

- removes points on the small subgroup
- permutes points in the large subgroup

Any multiple of h has the same effect.

In pairing-friendly curves, multiplying by a multiple of h is more efficient.

Idea

The scalar is written base p , so powers of p are performed applying the Frobenius map.

Encoding to Curve

Goal: A function $\{0, 1\}^* \rightarrow \mathbb{G}$ that is an encoding to the elliptic curve.

Algorithm:

1. $u = \text{HashToField}(M, \text{DST}, 1)$
2. $Q = \text{MapToCurve}(u)$
3. $P = \text{ClearCofactor}(Q)$

Part II – Design Considerations

Encoding vs Hash

Hash to Finite Field

Mapping to Curve

Clearing Cofactor

Hash to Curve

Hashing to
Curves:
From Theory to
RFC 9380
Specification

Armando Faz

Encoding vs Hash

Hash to Finite Field

Mapping to Curve

Clearing Cofactor

Hash to Curve

Can an encoding be directly used a hash?

The construction

$$H(m) = f(h(m))$$

where f is a mapping, and h is a standard hash function.

No

- The output of f does not reach all points in the curve.
- Some points are more likely to appear than others.
- Output is easily distinguishable from a uniformly random point.

Can scalar multiplication be used as a hash?

The construction

$$H(m) = h(m)G$$

where G is a generator of order q ,

$$h: \{0, 1\}^* \rightarrow \mathbb{Z}/n\mathbb{Z}.$$

No

- The discrete logarithm of the output is known.
- Make some protocols insecure, such as IBE.

What about a combination?

Brier et al. showed a construction for f be SWU or Icart's mapping

$$H(m) = f(h_1(m)) + h_2(m)g$$

where

g is generator of order n ,

$$h_1 : \{0, 1\}^* \rightarrow \mathbb{F}_p,$$

$$h_2 : \{0, 1\}^* \rightarrow \mathbb{Z}/n\mathbb{Z}.$$

Yes, but inefficient It needs to compute a scalar multiplication plus the mapping evaluation.

<https://eprint.iacr.org/2009/340>

Preferred Method: Encode Twice and Add

Brier et al. showed another construction for f is the Icart's mapping

$$H(m) = f(h_1(m)) + f(h_2(m))$$

where

$$h_1, h_2: \{0, 1\}^* \rightarrow \mathbb{F}_p.$$

Yes, it works

But, does this apply to mappings other than Icart's?

<https://eprint.iacr.org/2009/340>

Preferred Method: Generalization by FFSTV

Farashahi et al. showed a more general construction

$$H(m) = f(h_1(m)) + \dots + f(h_s(m))$$

where $s > \text{genus}(E)$, and f is well-distributed.

Bingo! The Encode-Twice-and-Add technique works for elliptic curves on more mappings.

<https://eprint.iacr.org/2010/539>

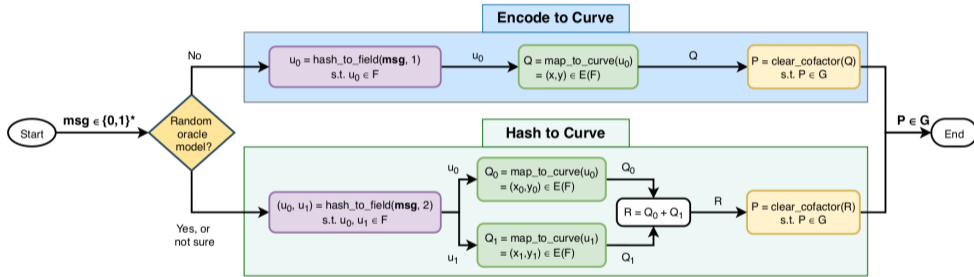
Hash to Curve

Goal: A function $\{0, 1\}^* \rightarrow \mathbb{G}$ that is a hash.

Strategy:

1. $\{u_1, u_2\} = \text{HashToField}(M, \text{DST}, 2)$
2. $Q_1 = \text{MapToCurve}(u_1)$
3. $Q_2 = \text{MapToCurve}(u_2)$
4. $Q = Q_1 + Q_2$
5. $P = \text{ClearCofactor}(Q)$

Framework for Hash to Curve



Part III

RFC 9380 Specification

RFC 9380

Part III – RFC 9380 Specification

RFC 9380

The Need for Secure Hash to Curve Functions

Before

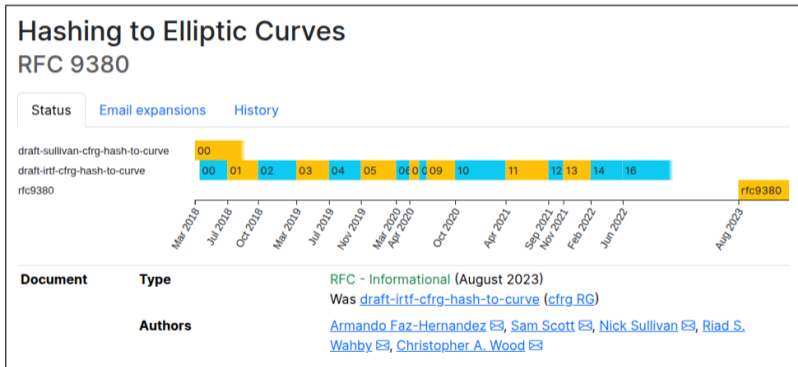
- No concrete guidance about how to perform hash to curve securely.
- Papers on the topic are difficult to grasp.
- Many results come from algebraic-geometry area.
- Insecure algorithms are popular.

Now

- Clear guidance about hashing to curve functions.
- P256, Curve25519, BLS12381, and more.
- Straight-line optimized implementations.
- Software libraries enable support.
- Protocol designers are using our methods.

RFC 9380 Timeline

It started in 2018 as an Internet draft at Crypto Forum Research Group (CFRG). Since 2023 it achieved RFC status (RFC 9380)



<https://datatracker.ietf.org/doc/rfc9380/>

A suite is a set of parameters and algorithms to define a particular instance.

- The curve, its parameters, and its main (sub)group.
- The hash or XOF function.
- The best mapping for the curve.
- Pseudo-code for optimized implementations.

Common Elliptic Curves:

- P-256, P-384, P-521.
- Curve25519, Curve448.
- edwards25519, edwards448.
- secp256k1.
- BLS12-381.
- Interoperability with quotient groups: ristretto and decaf.

Extensible: guidance for defining suites for other curves.

Reference Implementations

Repository: <https://github.com/cfrg/draft-irtf-cfrg-hash-to-curve>

All algorithms written in Sage/Python.

- Unit testing for encoding and hash to curve.
- Known-answers tests.
- More reference implementations in Go¹ and rust².

¹<https://github.com/armfazh/h2c-go-ref>

²<https://github.com/armfazh/h2c-rust-ref>

Compliant Implementations

- BoringSSL - C/C++
- CIRCL - Go
- libsodium - C/C++
- MIRACL Core - C/C++
- pairing-plus - rust
- RELIC - C/C++
- Apache Milagro Crypto Library - rust
- Zig's standard library - Zig

Part IV

Protocols

Protocols Using Hash To Curve

Part IV – Protocols

Protocols Using Hash To Curve

Oblivious PRFs (RFC 9497)






Two-party protocol to evaluate a pseudorandom function (PRF): $y = \text{PRF}_k(x)$

- Client holds x and Server has a key k .
- Client learns y and Server learns nothing.
- Oblivious: Client cannot learn k , and Server cannot learn x, y .

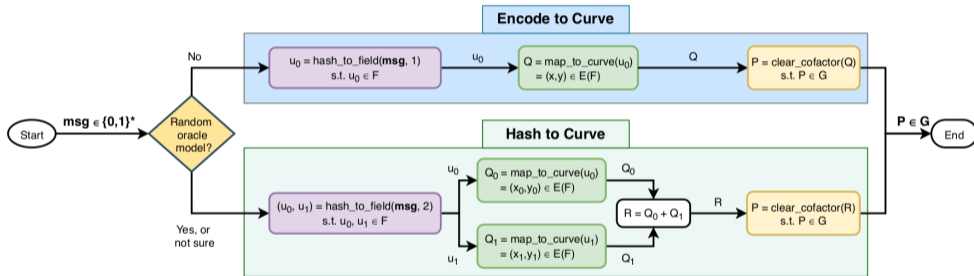
Protocol

Client(x)		Server(k)
$r \leftarrow_{\$} \{1, \dots, q\}$		
$P = rH(x)$	\xrightarrow{P}	
	\xleftarrow{Q}	$Q = kP$
$y = r^{-1}Q = kH(x)$		

Protocols

- Identity-based Encryption (IBE).
- Private Set Intersection (PSI).
- BLS Signatures 
- BBS Short Group Signatures 
- CPace, a balanced composable PAKE 
- ARC: Anonymous Rate-Limited Credentials 
- ACT: Anonymous Credit Tokens 
- VOPRF: Verifiable Oblivious Pseudorandom Functions (RFC 9497)
- VRF: Verifiable Random Functions (RFC 9381)
- SPAKE2, a Password-Authenticated Key Exchange (RFC 9382)
- OPAQUE: Augmented Password-Authenticated Key Exchange (RFC 9807)
- ⋮

Encoding and Hashing to Curve



Summary

- Concrete specification of hash to curve functions.
- Many issues: different curves, algorithms, full of subtleties.
- Constant-time and deterministic algorithms.
- RFC 9380 provides interoperability and best practices.
- Many protocols need a hash to curve function.

Thanks for your attention, and thanks the organizers for the invitation.

Contact Info:

- `ask-research@cloudflare.com`
- `rfc9380@ietf.org`
- `cfrg@ietf.org`