


ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ

An overview of the Signal Protocol

ΦΛΩΡΙΑΣ ΠΑΠΑΔΟΠΟΥΛΟΣ (Α.Ε.Μ.: 874)

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΔΗΜΗΤΡΙΟΣ ΠΟΥΛΑΚΗΣ

ΘΕΣΣΑΛΟΝΙΚΗ 2023

Abstract

This study was made for an assignment of the class "Elliptic-Curve Cryptography" of my master's program. For the assignment, the Signal Protocol (as described in [1, 2, 3, 4]), had to be studied, analyzed and represented in a more explanatory and student-friendly way.

Keywords: Elliptic curve Cryptography, Edwards Curves, Montgomery Curves, Curve25519, Curve448, EdDSA, ECDH, XEdDSA, X3DH protocol, Double Ratchet, Signal Protocol

Contents

Chapter 1

Preliminaries

In this chapter, we explore Elliptic Curve Cryptography (ECC) in depth, covering fundamental mathematical principles and practical applications. We start by introducing algebraic and projective curves over a field, explaining the Weierstrass equation for elliptic curves, and delving into point operations, including the addition law for points on elliptic curves and scalar multiplication.

We continue by exploring two important curve families: Edwards curves and Montgomery curves. Our goal is to highlight their "bi-rational equivalence" and define Edwards curves, twisted Edwards curves, and Montgomery curves. Additionally, we shall discuss the different addition laws for both curves.

In the context of the Signal Protocol, we will mention specific curves, Curve25519 and Curve448, which are utilized due to their computational efficiency and advantages in terms of implementation and resilience against side-channel attacks. These curves are used for Elliptic Curve Diffie-Hellman (ECDH) key exchange and the Edwards-curve Digital Signature Algorithm (EdDSA), a variant of the Elliptic Curve Digital Signature Algorithm (ECDSA).

In summary, a solid summary for the main concepts of ECC that are needed will be provided, covering key definitions, different forms of elliptic curves, and point operations, while also mentioning the importance of Curve25519 and Curve448 for the Signal Protocol.

1.1 Fundamentals of Elliptic Curve Cryptography

This section introduces the fundamentals of elliptic curves, including their definitions, different forms, and the operations performed on their points. Understanding these concepts is essential for comprehending the applications of ECC in modern cryptography.

1.1.1 Elliptic Curves

Definition 1.1. Let K be a field with $\bar{K}$ its algebraic closure[1] and $\mathbb{P}^n_K$ denote the projective space of dimension n over K. Moreover, let also $f(x,y) \in K[x,y] \backslash K$ and $F(x,y,z) \in K[x,y,z]$ be a homogeneous polynomial of degree ≥ 1. Then :

(a_1) An algebraic (plane) curve over K, defined by $f(x,y)$, is the set
$$V_f = \{(x,y) \in \bar{K}^2 / f(x,y) = 0\}$$

(a_2) If L is a field such that $K \subseteq L \subseteq \bar{K}$, the set of the L-rational points of V_f is the set
$$V_f(L) = \{(x,y) \in L^2 / f(x,y) = 0\}$$

(b_1) A projective algebraic (plane) curve over K, defined by $F(x,y,z)$, is the set
$$V_F = \{P \in \mathbb{P}^2_{\bar{K}} / f(P) = 0\}$$

[1] An algebraic closure is an algebraic extension of K that is algebraically closed. [5, 6]

(b_2) If L is a field such that $K \subseteq L \subseteq \bar{K}$, the set of the L-rational points of V_F is the set
$$V_F(L) = V_F \cap \mathbb{P}^2_L$$

We will continue by defining the well-known Weierstrass equation for elliptic curves:

Definition 1.2. A Weierstrass equation of an elliptic curve E over a field K is
$$E: \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$
where $a_1, a_2, a_3, a_4, a_6 \in K$ and $\Delta \neq 0$ where Δ denotes the discriminant of E.

Remarks.

1. Let $f(x, y) = y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6$, with $a_1, a_2, a_3, a_4, a_6 \in K$ and $\Delta \neq 0$. Then, we also say that V_f is an elliptic curve over K in Weierstrass form.
2. The condition $\Delta \neq 0$ ensures that V_E has no singular point. It can be proven that this can happen if-f $4a_4^3 + 27a_6^2 = 0$. For a detailed proof we refer the reader to [5].

With the above equation we have not given the definition of an elliptic curve[2] but only given an equation of this object. It should be known that an elliptic curve is an abstract object with many models, one of which is the Weierstrass equation. Others include the Edwards model and the Montgomery model, which we will define in the next section.

Moreover, we shall often confuse the definition of an elliptic curve and of its equation but one has to keep in mind that abstract curve $\neq$ a model of a curve $\neq$ an equation of the curve [7].

Next, we will mention a more complete version of the curve (in terms of Bézout Theorem[3]), as described by the projective Weierstrass equation:

Definition 1.3. A (projective) Weierstrass equation of an elliptic curve E over a field K is
$$\tilde{E}: \ y^2z + a_1xyz + a_3yz^2 = x^3 + a_2x^2z + a_4xz^2 + a_6z^3,$$
where $a_1, a_2, a_3, a_4, a_6 \in K$ and $\Delta \neq 0$ where Δ denotes the discriminant of E.

Remark. By defining the homogeneous polynomial $F(x, y, z)$ associated with f (i.e. such that $F(x, y, 1) = f(x, y)$), we can obtain the projective version of the affine curve in Definition 1.2. We can also say that V_F is an elliptic curve over K in Weierstrass form.

Additionally, we can observe that this version is more complete by distinguishing the points of V_F. Namely, we have the affine points of V_F (i.e. the ones with $z \neq 0$) and the points at infinity of V_F (i.e. the ones with $z = 0$). More precisely, by finding a representative $z = 1$, we can easily see that the affine points of V_F are the points of V_f. Also, letting $z = 0$ in the equation, we get $x^3 = 0$ and we see that the curve has a unique point at infinity, which we will denote $O = (0 : 1 : 0)$. Lastly, one can easily prove that the point O is non singular and therefore V_F is non-singular.

Finally, when $char(K) \neq 2, 3$, by using morphisms one can derive a simplified Weierstrass model of the form $y^2 = x^3 + ax + b$ (or $y^2z = x^3 + axz^2 + bz^3$ for the projective version), where $a, b \in K$ such that $4a^3 + 27b^2 \neq 0$. For more information the reader is referred to [5, 6].

1.1.2 Operations on points of elliptic curves

We can define[4] an addition law for points of elliptic curves over K of simplified Weierstrass form (with $char(K) \neq 2, 3$) as described below:

[2](Abstract definition) An elliptic curve over a field K is a projective non-singular curve of genus 1 with a K-rational point O. [5, 6]

[3]For more information, the reader is referred to [5, 6]

[4]For a more detailed description of the concept, [5, 6] are suggested.

Let $P, Q \in V_F$ be two points and E_{PQ} be the line connecting them (tangent to V_F if $P = Q$) and R be the third point of intersection of L with V_F by Bézout. Let L' be the line connecting R and O. Then, $P + Q$ is the residual point of intersection of L' and V_F.

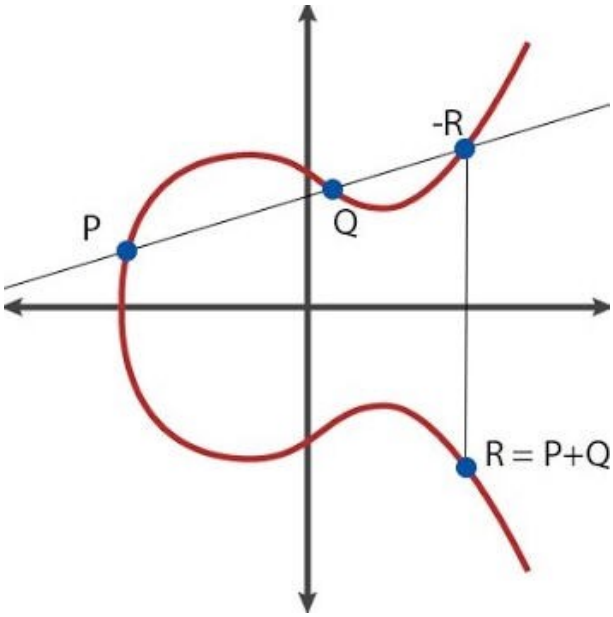


Figure 1.1: Example - Addition law for Weierstrass elliptic curves. [link]

It can be proven that the points of an elliptic curve form a commutative group under this addition law, with the point at infinity O as the identity element and the inverse of a point $P = (a : b : 1)$ being its symmetric about the x-axis, $-P = (a : -b : 1)$. Moreover, the proposition below can give us the coordinates of the point $P + Q$ when adding some P and Q:

Proposition 1.1. Let $P_i = (x_i : y_i : 1)$ $(i = 1, 2, 3)$ be points of an elliptic curve V_F in simplified Weierstrass form such that $P_1 + P_2 = P_3$. Then, $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = -\lambda x_3 - \mu$, where

- If $P_1 \neq P_2$, $\lambda = \dfrac{y_1 - y_2}{x_1 - x_2}$ and $\mu = y_1 - \lambda x_1$
- If $P_1 = P_2$, $\lambda = \dfrac{3x_1^2 + a}{2y_1}$ and $\mu = y_1 - \lambda x_1$

Finally, we can also define another operation with these points, the scalar multiplication:

$$nP = \underbrace{P + \ldots + P}_{\text{n times}}$$

1.2 Families of Elliptic Curves

Elliptic curve cryptography relies on two important families of curves: Edwards curves and Montgomery curves. Edwards curves, introduced by Harold M. Edwards, offer efficiency and security advantages. Montgomery curves, developed by Peter L. Montgomery, provide faster computations and enhanced resistance against side-channel attacks.

Despite their differences, these curve families are what we will define as "bi-rationally equivalent", allowing for seamless conversion between them. This section explores their definitions, properties, and conversion formulas, emphasizing their importance in cryptographic applications like the Signal Protocol.

1.2.1 Edwards Curves

One really important family of elliptic curves that is being used not only in parts of the Signal Protocol, but in elliptic curve cryptography in general, are Edwards curves and their generalization, the twisted Edwards curves.

Definition 1.4. Let K be a field with $char(K) \neq 2$. Then

(a) An Edwards curve is a curve $V_E(K)$ defined by $E : x^2 + y^2 = 1 + dx^2y^2$ where $d \notin \{0, 1\}$.

(b) A twisted Edwards curve is a curve $V_E(K)$ defined by $E : ax^2 + y^2 = 1 + dx^2y^2$ where $a \neq d$ and $a, d \neq 0$.

Remarks.

1. Harold M. Edwards first introduced these curves with a slightly less general formula in [8]. The above form of Edwards curves was first used in [9], where Daniel J. Bernstein and Tanja Lange proved their efficiency for cryptographic applications.
2. Edwards curves constitute a specific case of twisted Edwards curves (precisely for $a = 1$).
3. From the above formulas, we can compute the respective projective Edwards equations by defining the homogeneous polynomial, as we did in Weierstrass form in the previous section.

Next, we would like to state an important theorem connecting Edwards curves with elliptic curves. For this, we will need the definition below:

Definition 1.5. Two curves are said to be birationally equivalent if there exist maps between the curves given by fractions of polynomials (called rational maps) which map almost all points between the curves and which are compatible with the group law. Namely, $\phi_1 : E_1 \mapsto E_2$ and $\phi_2 : E_2 \mapsto E_1$, $\phi_i(P + Q) = \phi_i(P) + \phi_i(Q)$, where ϕ_i are rational maps for all $P, Q, P + Q$ where ϕ_i is defined on E_i.

It can be proven that every elliptic curve over a non-binary field is birationally equivalent to a curve in Edwards form over an extension of the field, and in many cases over the original field. More specifically, for finite fields which are our main interest, the following theorem holds true (proof in [9]:

Theorem 1.1. Let K be a finite field with $char(K) \neq 2$ and let E be an elliptic curve over K s.t. $V_E(K)$ has an element of order 4 and a unique element of order 2, then there is a non-square $d \in K$ such that the curve $x^2 + y^2 = 1 + dx^2y^2$ is birationally equivalent to E over K.

For cryptographic uses, we prefer to use a special class of twisted Edwards curves in which a is a square and d is non-square in K. These are called K-complete or simply complete and for such a curve E, we can define an addition law on the group $V_E(K)$ of its K-rational points, as described below:

The neutral element is $(0, 1)$ (while on the Weierstrass form it was the point at infinity) and the negative of a point (x, y) is $(-x, y)$. Also, the sum of two points (x_1, y_1) and (x_2, y_2) is defined as

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_1 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2} \right).$$

One interesting feature of this additional law is that it is complete. This means that the formulas work for all pairs of input points on the curve, with no exceptions for doubling, no exceptions for the neutral element, no exceptions for negatives, etc.

This information on Edwards curves and twisted Edwards curves is enough for our later purposes where we will also see how they constitute an efficient model of elliptic curves for use in cryptography and how they can be useful for the Signal Protocol.

1.2.2 Montgomery Curves

The Montgomery curves, introduced by Peter L. Montgomery in 1987, constitute a different model of elliptic curves which are used for certain "faster" computations and in cryptography applications, as well as in the Signal Protocol.

Definition 1.6. Let K be a field with $char(K) \neq 2$ and let $A, B \in K$ s.t. $B \neq 0$ and $A \neq \pm 2$. Then, the curve defined by the equation $Bv^2 = u^3 + Au^2 + u$ is a Montgomery curve.

It can be easily proved [10] that a Montgomery curve $Bv^2 = u^3 + Au^2 + u$ has the form[5] of a Weierstrass curve $a_0y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$, where $(a_0, a_1, a_3, a_2, a_4, a_6) = (B, 0, 0, A, 1, 0)$ and is geometrically non-singular, so it is a Weierstrass curve. Therefore, it is evident that the addition law defined for this type of curves will be similar to the one for Weierstrass curves. More specifically:

The neutral element is the point at infinity and the inverse of a point $P = (a : b : 1)$ is, $-P = (a : -b : 1)$. Moreover, the proposition below can give us the coordinates of the point $P + Q$ when adding some P and Q:

Proposition 1.2. Let $P_i = (u_i : v_i : 1)$ $(i = 1, 2, 3)$ be points of an elliptic curve V_F in Montgomery form such that $P_1 + P_2 = P_3$. Then, $u_3 = B\lambda^2 - A - u_1 - u_2$ and $v_3 = -\lambda u_3 - \mu$, where

- If $P_1 \neq P_2$, $\lambda = \dfrac{v_1 - v_2}{u_1 - u_2}$ and $\mu = v_1 - \lambda u_1$
- If $P_1 = P_2$, $\lambda = \dfrac{3u_1^2 + 2Au_1 + 1}{2Bv_1}$ and $\mu = v_1 - \lambda u_1$

Finally, we should mention the existence of the Montgomery ladder algorithm which can efficiently compute scalar multiples on Montgomery curves. This algorithm is not only fast but is also resistant to side-channel attacks like timing attacks, which makes it an excellent candidate for use in real-world applications of cryptography, like the Signal Protocol. Moreover, it can work only using the x-coordinate of a point and calculating the y-coordinate once in the end of the algorithm. While we won't explore it in depth here, more information can be found in references like [10] and [11].

Remark. Montgomery curves are widely used in elliptic curve cryptography due to their efficiency and security features. They enable fast arithmetic operations, especially scalar multiplication, resulting in accelerated cryptographic computations. Additionally, Montgomery curves offer resistance against side-channel attacks like timing attacks and power analysis. This is because they can facilitate constant-time implementation, crucial for preventing timing attacks. For more information on Montgomery curves the interested reader can refer to [11].

1.2.3 Elliptic curve conversions

As we mentioned before, both twisted Edwards and Montgomery curves are useful for the Signal Protocol, each for their respective unique features. However, what we haven't mentioned yet is that these types of curves are bi-rationally equivalent. This means that by establishing the appropriate mappings between the two forms, we can effectively leverage the distinct advantages of each one without needing to maintain separate curves. Consequently, in the Signal Protocol, we observe a consistent use of these curve types, seamlessly transitioning between their respective forms whenever necessary.

[5]The conventional definition of a Weierstrass curve in literature typically assumes that $a_0 = 1$. However, by relaxing this restriction, we open up the possibility of utilizing Montgomery curves without needing any rescaling and introducing merely a nuanced layer of complexity to the theory of Weierstrass curves, with this adjustment.

In particular, the transformation formulas from the twisted Edwards curve $ax^2 + y^2 = 1 + dx^2y^2$ to the Montgomery curve $Bv^2 = u^3 + Au^2 + u$ (with the respective restrictions for their parameters) are

$$u = \frac{1+y}{1-y} \text{ and } v = \frac{1+y}{x(1-y)} = \frac{u}{x}$$

where the parameters are connected by $A = 2\frac{a+d}{a-d}$ and $B = \frac{4}{a-d}$. Correspondingly, the formulas from the Montgomery to the twisted Edward curve are

$$x = \frac{u}{v} \text{ and } y = \frac{u-1}{u+1}$$

where the parameters are connected by $a = \frac{A+2}{B}$ and $d = \frac{A-2}{B}$.

By handling the point to infinity carefully as input and output, the above mappings can be used for all values necessary, as shown in [10] and [12].

1.3 Safe Curves

In the context of the Signal Protocol, two specific curves play a pivotal role: Curve25519 and Curve448. These standardized curves, endorsed by the Internet Research Task Force (IRTF) [13], offer distinctive advantages in elliptic curve cryptography.

Curve25519 and Curve448 belong to the family of Montgomery curves, defined by the equation $v^2 = u^3 + Au^2 + u$. As we mentioned before, these types of curves exhibit desirable properties, including efficient constant-time implementation and resilience against various side-channel attacks like timing and cache attacks. Additionally, these curves have their bi-rationally equivalent Edwards curves counterparts which feature the fastest known complete formulas for elliptic-curve group operations. Specifically, the Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ is utilized for primes satisfying $p = 3\, mod 4$, while the twisted Edwards curve $-x^2 + y^2 = 1 + dx^2y^2$ is employed when $p = 1\, mod 4$.

By precisely incorporating Curve25519 and Curve448, the Signal Protocol leverages the distinct advantages and computational efficiency of these curves, ensuring secure and efficient elliptic curve operations tailored to the protocol's cryptographic needs.

Furthermore to that, we shall elaborate on the exact nature of these curves and their corresponding elliptic curve Diffie-Hellman[6] functions, named "X25519" and "X448" accordingly. However, those curves are not only used for ECDH, but also for a variant of the Elliptic Curve Digital Signature Algorithm[7] (ECDSA), the Edwards-curve Digital Signature Algorithm (EdDSA) as we will see in the main chapter. For our purposes, we will only highlight that the main difference of EdDSA and ECDSA is that EdDSA uses twisted Edwards curves rather than Weierstrass curves; for more details on EdDSA we refer the interested reader to [14] and [15].

· **Notation and definitions**

Definition 1.7. Let V_E be an elliptic curve with $|V_E| = n$ points and G be a generator point for a subgroup of prime order q. Then, the parameter c that represents the ratio between n and q is called a cofactor. Mathematically, it is defined as $c = n/q$.

Remark. When the cofactor is not 1 then the subgroup of prime order is a strict subset of the curve. When considering a point, verifying that the curve coordinates match the curve equation is not sufficient to guarantee that the point is on the appropriate subgroup. Moreover, there will

[6]We assume familiarity with Elliptic Curve Diffie-Hellman (ECDH) key agreement protocol. However, for those who are not acquainted, please refer to [5] for further details.

[7]We assume familiarity with ECDSA too. For beginners, please refer to [5] for further information.

be points whose order is not a multiple of the order of the subgroup. This is what happens, for instance, with Curve25519, which has a cofactor of 8. Such curves require some extra care in the protocol that uses them. For instance, when doing an elliptic curve Diffie-Hellman key exchange over Curve25519, the Diffie-Hellman private keys must be chosen as multiples of 8 (setting the three least significant bits to zero), ensuring that the points will be in the proper subgroup.

Another important concept is that of the encoding of points and their "bitlength".
In particular, in Elliptic Curve Cryptography (ECC), the bitlength for an encoded point refers to the number of bits required to represent the coordinates of a point on the curve in an encoded form and is used when transmitting or storing elliptic curve points. For example, usually if the field prime p has a bitlength of $|p|$, and the encoding format requires 8 bits for each coordinate, then the bitlength for an encoded point would be: $b = 8\lceil(|p|+1)/8\rceil$

Finally, we will use the notation below when defining the exact parameters of the curves:

Notation	Description
p	Denotes the prime number defining the underlying field.
$\mathbb{F}_p$	The finite field with p elements.
A	Montgomery curve constant
d	Twisted Edwards curve constant
B	A generator point defined over $\mathbb{F}_p$ of prime order.
q	The order of the prime-order subgroup.
c	The cofactor corresponding to the generator point.
b	The bitlength for an encoded point or integer.
$U(B)$	The u-coordinate of the elliptic curve point B on a Montgomery curve.
$V(B)$	The v-coordinate of the elliptic curve point B on a Montgomery curve.
$X(B)$	The x-coordinate of the elliptic curve point B on a (twisted) Edwards curve.
$Y(B)$	The y-coordinate of the elliptic curve point B on a (twisted) Edwards curve.
u, v	Coordinates on a Montgomery curve.
x, y	Coordinates on a (twisted) Edwards curve.

1.3.1 Curve25519

For the 128-bit security level[8], the prime $p = 2^{255} - 19$ is recommended for by IRTF [13] when using Curve25519. Within the confines of our study, we shall only state the parameters with which the curve is used in the protocols mentioned above. For further details, consult [16].

In the table below we state the parameters of the "Curve25519" as a Montgomery curve $v^2 = u^3 + Au^2 + u$, as well as those of its birationally equivalent twisted Edwards curve $-x^2 + y^2 = 1 + dx^2y^2$, called "Edwards25519".

Moreover, as stated in [13], their bi-rational maps are:

$$(u, v) = \left(\frac{1+y}{1-y}, \sqrt{-486664} \cdot \frac{u}{x}\right) \quad \text{and} \quad (x, y) = \left(\sqrt{-486664} \cdot \frac{u}{v}, \frac{u-1}{u+1}\right)$$

[8] A 128-bit security level in elliptic curve based asymmetric encryption and digital signature schemes requires around 2^{128} operations to solve the underlying discrete logarithm problem or break the security.

Par.	Value
p	$2^{255} - 19$
A	486662
d	37095705934669439343138083508754565189542113879843219016388785533085940283555
$U(B)$	9
$V(B)$	14781619447589544791020593568409986887264606134616475288964881837755586237401
$X(B)$	15112221349535400772501151409588531511454012693041857206046113283949847762202
$Y(B)$	46316835694926478169428394003475163141307993866256225615783033603165251855960
q	$2^{252} + 27742317777372353535851937790883648493$
c	8
b	256

Table 1.1: Curve25519 Parameters

The Montgomery curve defined here is equal to the one defined in [16], and the equivalent twisted Edwards curve is equal to the one defined in [14].

Concluding, Curve25519's widespread acceptance and comprehensive study within the cryptographic community validate its suitability for secure and reliable elliptic curve cryptography applications. For further details, please refer to the specified sources..

1.3.2 Curve448

For achieving a higher level of security, the prime $p = 2^{448} - 2^{224} - 1$ is recommended for the 224-bit security level when using Curve448. For our purposes we shall only mention the parameters of its Montgomery form and of its bi-rationally equivalent twisted Edwards counterpart.

In the table below we state the parameters of the "Curve448" as a Montgomery curve $v^2 = u^3 + Au^2 + u$, as well as the parameters of its birationally equivalent twisted Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ (which is not called "Edwards448", more info on IRTF's report).

Param.	Value
p	$2^{448} - 2^{224} - 1$
A	156326
d	611975850744529176160423220965553317543219696871016626328968936415087860042636474891785599283666020414768678979989378147065462815545017
$U(B)$	5
$V(B)$	355293926785568175264127502063783334808976399387714271831880898435169088786967410002932673765864550910142774147268105838985595290606362
$X(B)$	193744608242252278496651022980508834618059227425397611426356190235586356663831269322437834903366616154771054188245590407719440490631313325347786581146764911381806680989511535200738674851542688033669247488217860989454750388541
$Y(B)$	4
q	$2^{446} - 138180668098951153520073867485154268803366924748821786098945475038854$
c	4
b	456

Table 1.2: Curve448 Parameters

Moreover, the bi-rational maps between them are:

$$(u, v) = \left(\frac{y-1}{y+1}, \sqrt{156324} \cdot \frac{u}{x}\right) \quad \text{and} \quad (x, y) = \left(\sqrt{156324} \cdot \frac{u}{v}, \frac{1+u}{1-u}\right)$$

For further details on the curves and mappings defined here, please refer to [17].

Concluding, as the IRTF states in [13]:

"The 224-bit security level of curve448 is a trade-off between performance and paranoia." Reasonable projections of the abilities of classical computers conclude that curve25519 is perfectly safe with its 128-bit security. However, for designs with relaxed performance requirements and a desire to mitigate potential advances in elliptic curve attacks, Curve448 is also available. All in all, by offering both curves for use, varying cryptographic applications can be accommodated, balancing performance and security considerations.

Chapter 2

The Signal Protocol

This chapter examines the Signal Protocol, focusing on its key elements and mechanisms as described in [1, 2, 3, 4]. We analyze the XEdDSA and VXEdDSA signature schemes, the X3DH Key Agreement Protocol for secure key exchange, and the Double Ratchet Algorithm for forward secrecy and message integrity. The goal is to provide a concise overview of the Signal Protocol and its cryptographic foundations, highlighting their importance in achieving secure and private communication.

Additionally, we should note that our overview will not include details on VXEdDSA or anything regarding the Sesame algorithm. Interested readers are encouraged to consult the mentioned sources for further information on these.

2.1 XEdDSA & VXEdDSA Signature Schemes

Within the section, we will explain in detail the concepts that are covered on the first paper [1] of the official documentation of the Signal Protocol, mainly the XEdDSA Signature Scheme and the algorithmic functions necessary to define this and other schemes.

The paper provides instructions on creating and validating EdDSA-compatible signatures using the public key and private key formats initially established for the X25519 and X448 elliptic curve Diffie-Hellman functions ([13, 14, 15, 16]). We refer to this signature scheme as "XEdDSA" in general and "XEd25519" or "XEd448" when it is used with Curve25519 and Curve448, respectively. XEdDSA offers the convenience of employing a single key pair format for both elliptic curve Diffie-Hellman and signatures, enabling the possibility of using the same key pair for both algorithms in certain scenarios.

Additionally, this document introduces "VXEdDSA," an extension of XEdDSA that transforms it into a verifiable random function (VRF). For the objectives we have in mind, we will not delve deeper into VXEdDSA but, the curious reader can look up [18, 19].

2.1.1 Algorithmic functions

We will start off with a table containing the description of several important symbols in order to make the pseudocode of the paper more understood.

· **Elliptic curve conversion functions**

As we have briefly mentioned before, XEdDSA and VXEdDSA are defined for twisted Edwards curves ((x, y) for their points), while ECDH uses Montgomery curves ((u, v) for their points), whose scalars are often calculated using the Montgomery ladder algorithm.

On one side, in the case of EdDSA signatures, a public key takes the form of a compressed point, comprising a twisted Edwards y-coordinate and a sign bit (s) that can be either 0 or 1.

Notation	Description
`a/b (modp)`	This is calculated as $ab^{-1}\, modp$.
`inv(a) (modp)`	This returns $a^{-1}\, modp$ when $a \neq 0$ and 0 when $a = 0$.
`ceil()`	It rounds up to the nearest integer
`floor()`	It rounds down to the nearest integer.
`bytes_equal(X,Y)`	It is used to check if two byte sequences X and Y are equal.
`on_curve(P)`	It returns true if a point P satisfies the curve equation of the chosen curve.

Table 2.1: Description of basic notation and symbols used in [1].

This serves as an alternative way to represent a twisted Edwards point while the x-coordinate of the point can be computed easily from y using the the equation of the chosen curve.

On the other side, the Montgomery ladder further offers the advantage of accommodating each party's public key as a Montgomery u-coordinate, an approach that reduces the size of public keys without the need for point decompression.

Finally, we note that the integers are represented by b bits, encoded in little-endian form, where the first $b-1$ bits are used for the integer and the final bit by the sign. In the following pseudocodes, for a point $P = (x, y)$, the corresponding parts for the integer and the sign will be denoted as $P.y$ and $P.s$ respectively.

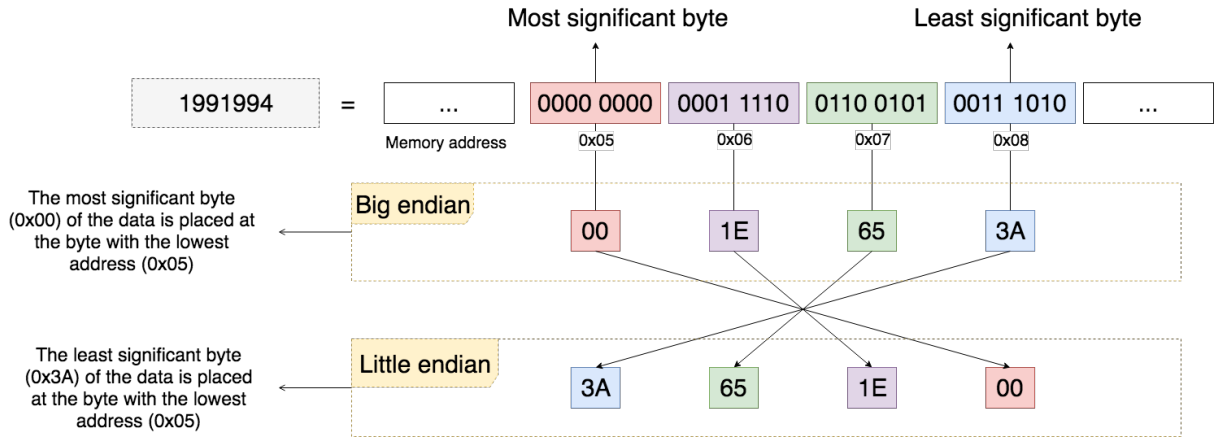


Figure 2.1: Representation of endian formats (Picture taken from [link])

From the above, it is evident that we will first need a function that converts a point (u, v) on the Montgomery curve, to its respective counterpart (x, y) of the bi-rationally equivalent twisted Edwards curve. More precisely, we want a function that takes as input u and returns the respective integer of the y-coordinate, as well as its proper sign bit. To do this, the function `convert_mont()` is used. As a subroutine it uses the function `u_to_y()`, which transforms the u-coordinate of a point on the Montgomery curve into the y-coordinate of the equivalent point on the twisted Edwards curve, by employing a bi-rational map specific to the curve chosen.

```
convert_mont(u):
  u_masked = u (mod 2|p|)
  P.y = u_to_y(u_masked)
  P.s = 0
  return P
```

However, the `convert_mont()` function, as it stands, does not consider the Montgomery curve's v coordinate. Consequently, it cannot differentiate between the two potential options for the twisted Edwards sign bit, which can create problems for our setup. This limitation is addressed by [20], who proposes the approach of setting the sign bit to zero.

The adjustments outlined in the literature, particularly the modification proposed in [20]

to set the sign bit to zero during the conversion process, address the potential ambiguity in distinguishing between different sign bit options. This adjustment is crucial for maintaining the integrity and security of the converted points and their corresponding keys.

In order to ensure compatibility with the conversion process, we define a twisted Edwards private key, represented by a scalar a, such that the corresponding twisted Edwards public key $A = aB$ has a sign bit of zero. Here, B refers to the twisted Edwards chosen generator point. On the other hand, a Montgomery private key (k) can be any scalar value.

```
calculate_key_pair(k):
  E = kB
  A.y = E.y
  A.s = 0
  if E.s == 1:
    a = -k (mod q)
  else:
    a = k (mod q)
  return A, a
```

To convert a Montgomery private key k into a twisted Edwards public-private key pair (A, a), we employ the `calculate_key_pair()` function. This function performs scalar multiplication of the Montgomery private key k with B. Subsequently, if necessary, it adjusts the private key a to ensure a sign bit of zero, following the methodology outlined in [20].

· **Hashing**

Finally, in order for the XEdDSA and VXEdDSA cryptographic schemes to function effectively, the utilization of a hash function becomes imperative. The standard hash function used by the Signal Protocol is SHA-512 [21].

In the paper `hash` is defined as a function that applies cryptographic hashing to a sequence X of input bytes and outputs an integer obtained by parsing the output of the cryptographic hash in little-endian format. Also, a family of hash functions $\{\texttt{hash}_\texttt{i}\}$ is defined, indexed by $i = 0, 1, \ldots$ such that $2^{|p|} - 1 - i > p$, where p and b are the curves constants.

```
hash_i(X):
  return hash(2^b - 1 - i || X)
```

Different $\texttt{hash}_\texttt{i}$ will be used for different purposes, to provide cryptographic domain separation. Therefore, this way different applications or purposes using cryptographic functions do not interfere with each other's security guarantees or produce unintended cryptographic relationships. Note also that $\texttt{hash}_\texttt{i}$ will never call hash with the first b bits encoding a valid scalar or elliptic curve point, since the first $|p|$ bits encode an integer greater than p.

Example 2.1. We will illustrate the operations of `hash`, $\texttt{hash}_0$, and $\texttt{hash}_1$ using SHA-256 as the hash function. The byte sequence "SECRET" will be hashed, and a value of $b = 8$ is assumed.

1. The `hash` of "SECRET" using SHA-256 is:
   $$0917b13a9091915d54b6336f45909539cce452b3661b21f386418a257883b30a$$
2. We first remind that the hexadecimal form of 255 is FF, notably $(2^8 - 1)_{10} = (FF)_{16}$. Hence, we compute, $\texttt{hash}_0(\text{SECRET}) = \texttt{hash}(2^8 - 1 \,||\, \text{SECRET}) = \texttt{hash}(\text{FF} \,||\, \text{SECRET})$. The SHA-256 hash of "FFSECRET" is:
   $$b8320373da12156a44bff90a1302a8e5578bf7c423239f89cf4bbc6a3f59c119$$
3. In the same way, for $\texttt{hash}_1$, we compute the SHA-256 hash of "FESECRET":
   $$625348bce4960651c4953bee07a1b4fa15263fea5c4ae255c1701e7149dbcfed$$

2.1.2 XEdDSA & VXEdDSA

In this segment, we will provide an elaborate explanation of the key components of XEdDSA but not VXEdDSA, as mentioned previously. Let's begin by acknowledging a key principle of XEdDSA: it relies on utilizing key pairs generated through the ECDH function X25519 (or

X448). To ensure compatibility with EdDSA and enable the creation and verification of EdDSA-compatible signatures, these key pairs first undergo a transformation process. This transformation process is build into the signature generation process as its first step and it is done using the `calculate_key_pair()` function.

· **Signature generation**

Subsequently, the normal steps for EdDSA signing are followed, as presented in [14]. All of the aforementioned is contained in the `xeddsa_sign()` function, which has as an input a tuple (k, M, Z), where:

- $k \doteqdot$ Montgomery private key (integer $modq$)
- $M \doteqdot$ Message to sign (byte sequence)
- $Z \doteqdot$ 64 bytes secure random data (byte sequence)

and as an output a signature $(R \,||\, s)$ (byte sequence of $2b$ bits), where R encodes a point and s an integer $modq$.

```
xeddsa_sign(k, M, Z):
  A, a = calculate_key_pair(k)
  r = hash_1(a || M || Z) (mod q)
  R = rB
  h = hash(R || A || M) (mod q)
  s = r + ha (mod q)
  return R || s
```

· **Signature verification**

For the verification function `xeddsa_verify()` an input $(u, M, R \,||\, s)$ is required, where:

- $u \doteqdot$ Montgomery public key (byte sequence of b bits)
- $M \doteqdot$ Message to verify (byte sequence)
- $R \,||\, s \doteqdot$ Signature to verify (byte sequence of $2b$ bits)

and as an output "true" or "false", depending on the results of the verification.

```
xeddsa_verify(u, M, (R || s)):
  if u >= p or R.y >= 2|p| or s >= 2|q|:
    return false
  A = convert_mont(u)
  if not on_curve(A):
    return false
  h = hash(R || A || M) (mod q)
  R_check = sB - hA
  if bytes_equal(R, R_check):
    return true
  return false
```

Lastly, we should note that in our desired scope we will not delve deeper into XedDSA and therefore, for performance and security analysis of XEdDSA, as well as more information about VXEdDSA, the interested reader is again referred to the primary source of this section, [1].

2.2 X3DH Key Agreement Protocol

X3DH, short for Extended Triple Diffie-Hellman, is a cryptographic protocol designed to establish secure end-to-end communication in various messaging applications, which was first introduced in [2]. By building upon the Diffie-Hellman key exchange, it offers a secure and efficient method for secure communication establishment. It incorporates forward secrecy[1] and cryptographic deniability[2].

This protocol is particularly suited for asynchronous settings, where one user has published information to a server while offline and another user can use that information to send encrypted data to the first and establish a shared secret key for future communication. The table below outlines the roles assigned to users and servers in the system.

User	Objective
Alice	Send initial encrypted data to Bob and establish a shared secret key.
Bob	Allow parties to establish a shared key with him and receive encrypted data. To accommodate Bob's offline status during Alice's attempts, a server may be employed to facilitate the process.
Server	Storing messages from Alice to Bob and providing Bob's published data to parties like Alice. A single server is assumed for simplicity, but in practice, multiple entities could fulfill the server role.

Table 2.2: Parties and their roles in X3DH

Additionally, for an application utilising X3DH, the following parameters should be selected:

Param.	Description
`curve`	X25519 or X448
`hash`	256 or 512-bit hash function, for example SHA-256 or SHA-512
`info`	ASCII string identifying the application
`Encode(PK)`	Function for encoding an X25519 or X448 public key (PK) into a byte sequence. More on this on [1].

Finally, before analyzing the X3DH setup, let's review the inner workings of X25519 and X448 functions (referenced in [13]). As an example, let's consider two users with private keys a and b, aiming to establish a shared secret using Curve25519.

Firstly, we remind that for ECDH, the Montgomery form is used for calculations and, especially the u-coordinate. Hence, from the public generator point B, only $U(B) = 9$ is necessary for the creation of the public keys, as described in Subsection 1.3. Subsequently, the above are used to compute the shared secret as illustrated in the figure above.

2.2.1 X3DH Setup

Let's explore the inner workings of X3DH, starting with initial key pairs and progressing to essential elements like the KDF and AEAD. This step-by-step exploration provides a comprehensive understanding of the protocol's mechanisms, preparing us for an in-depth analysis of the protocol itself in the next segment.

[1]Forward secrecy ensures the confidentiality of past communications, even if long-term secret keys are compromised in the future.

[2]Cryptographic deniability enables participants to deny their involvement or the communication content, providing plausible deniability. More details can be found in [22].

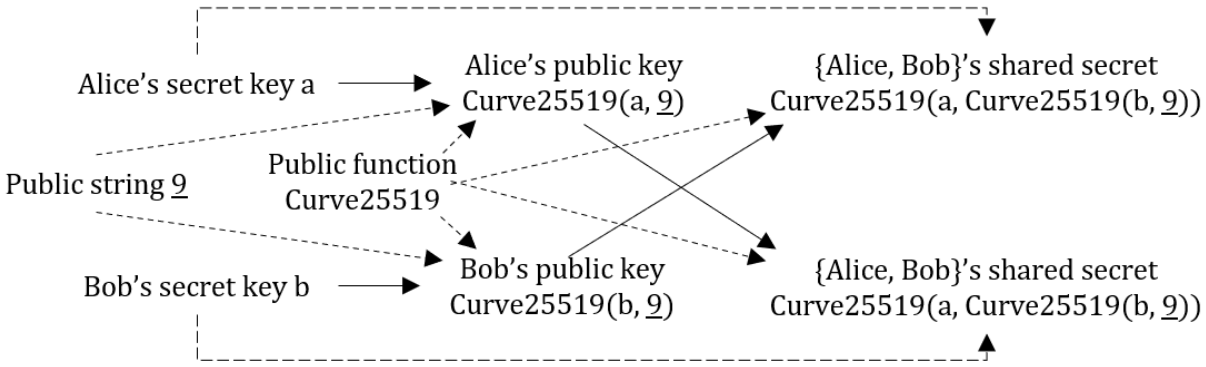


Figure 2.2: Data flow from secret keys through public keys to a shared secret with X25519 [16].

· **Key Derivation Function (KDF)**

In the context of X3DH, the Key Derivation Function (KDF) and HKDF (HMAC-based Key Derivation Function) play critical roles in deriving session keys and ensuring the confidentiality, integrity, and authenticity of exchanged information.

Definition 2.1. A key derivation function (KDF) is responsible for deriving one or more cryptographically strong secret keys from initial keying material. In order for its output to be strong, several security measures have to be considered, such as:

- The KDF should consistently produce the same output length regardless of the input.
- It should process all bits of the input and incorporate them into the output.
- To derive different outputs for the same input, a salt should be employed.
- To counteract brute-force attacks, the KDF should deliberately operate slowly.
- Regular updates and patches should be applied to address any vulnerabilities.

Definition 2.2. A Message Authentication Code (MAC) is used to verify the authenticity and integrity of a message. HMAC (Hash-based Message Authentication Code) is a specific type of MAC that combines a hash function with a secret key, providing enhanced security for message authentication.

HKDF (HMAC-based Key Derivation Function) operates based on the "extract-then-expand" principle, comprising two modules within its logical structure:

- In the initial stage, the KDF extracts a fixed-length pseudorandom key, K, from the input keying material.
- Subsequently, the second stage expands this key, generating multiple additional pseudorandom keys as the output of the KDF.

In some cases, input keying material may lack uniform distribution, enabling attackers to have partial knowledge or control. The "extract" stage aims to consolidate dispersed entropy into a strong pseudorandom key. The subsequent "expand" stage then extends the pseudorandom key to the desired length, generating additional keys as needed.

In the Signal Protocol, the KDF function takes the following inputs [23]:

- **HKDF *input keying material***: A byte sequence $F||KM$, where KM is an input containing secret key material, and F is a byte sequence of 32 0xFF or 57 0xFF bytes for X25519 and X448, respectively. The F is used to provide cryptographic domain separation.
- **HKDF *salt***: A zero-filled byte sequence with a length equal to the hash output length.
- **HKDF *info***: Additional information in the form of an ASCII string.

[More detailed information on KDF and HKDF can be found in [23, 24, 25].]

· **Authenticated Encryption with Associated Data**

AEAD (Authenticated Encryption with Associated Data) is a cryptographic solution that enables the secure transmission of both an encrypted message and associated data (AD). AD, which can be in plaintext form, is authenticated along with the message to ensure its integrity and authenticity. AEAD provides confidentiality, integrity, and authenticity for both the message and the associated data, ensuring their privacy and trustworthiness. Additional detailed information on AEAD can be found in [26].

· **Keys**

We will now present the key pairs used in the protocol, which should be in the format defined by the `curve` chosen. These public key - secret key pairs are:

- **Identity keys**: Long-term, bound to the device of the user.
- **Ephemeral keys**: Randomly generated, used only once for each run of the protocol.
- **Signed prekeys**[3]: Signed using the identity keys. Changed periodically on the server.
- **One-time prekeys**: Can be used (optionally) only one time in each X3DH run.

For the keys listed above, we will utilize the notation described in the following table. For simplicity, the table only presents the public keys, while acknowledging that each key pair consists of a corresponding private key.

Public key	Definition
IK_A	Alice's identity key
EK_A	Alice's ephemeral key
IK_B	Bob's identity key
SPK_B	Bob's signed prekey
OPK_B	Bob's one-time prekey

Finally, we should note that the 32-byte secret key created after a successful protocol run will be denoted as "SK" and it may be used in some post-X3DH secure communication protocol, like the Double Ratchet Algorithm, which we will define on the next section.

· **X3DH Notation**

Finally, the following notation is used in the pseudocode:

Notation	Description
`DH(PK1, PK2)`	Shared secret output from ECDH function using public keys `PK1` and `PK2`.
`Sig(PK,M)`	XEdDSA signature on byte sequence `M` with verification using public key `PK`.
`KDF(KM)`	32 bytes of output from the HKDF algorithm with specific inputs.

2.2.2 X3DH Protocol

The X3DH Protocol consists of three phases:

- **Phase 1**: Bob publishes IK_B, SPK_B and (optional) OPK_B.
- **Phase 2**: Alice gets a "prekey bundle" from the server to send an initial message to Bob.
- **Phase 3**: Bob receives and processes Alice's initial message.

The figure below further illustrates the process and the subsequent segments provide detailed explanations of each phase.

[3]Prekeys are keys that were uploaded to the server prior to any communication with the other party.

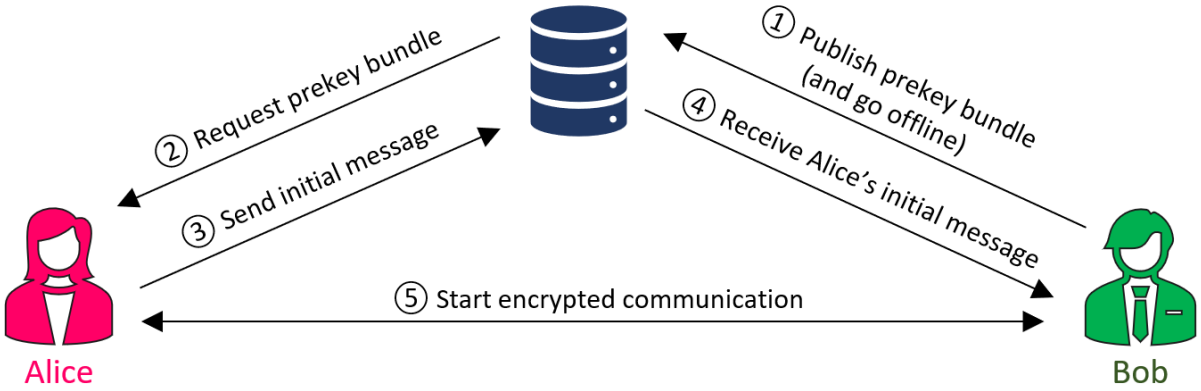


Figure 2.3: High-level diagram of X3DH [27].

· **Phase 1 (Publishing keys - Bob)**

Step 1. Bob provides the server with a set of elliptic curve public keys, including:

- Bob's identity key, IK_B
- Bob's signed prekey, SPK_B and signature, $\texttt{Sig}(IK_B, \texttt{Encode}(SPK_B))$
- A set of Bob's one-time prekeys $\{OPK_B^1, OPK_B^2, OPK_B^3, \dots\}$

Concerning the publication and storing of the public keys, the following should be kept in mind:

- Bob doesn't need to upload his identity key to the server more than once.
- Bob can upload new one-time prekeys as needed, especially when the server's supply of one-time prekeys is running low.
- Bob periodically uploads a new signed prekey and prekey signature, replacing the previous values. The old private key corresponding to the previous signed prekey is kept for a certain period to handle delayed messages. Eventually, Bob deletes this private key to ensure forward secrecy.

· **Phase 2 (Sending the initial message - Alice)**

In order for Alice to do a X3DH handshake with Bob, she performs the following:

Step 1. Alice reaches out to the server and retrieves a "prekey bundle" containing:

- Bob's identity key, IK_B
- Bob's signed prekey, SPK_B and signature, $\texttt{Sig}(IK_B, \texttt{Encode}(SPK_B))$
- (Optionally) One of Bob's one-time prekeys, denoted as OPK_B

Step 2. Alice verifies the prekey signature and aborts if verification fails.

Step 3. Alice generates an ephemeral key pair with public key EK_A.

Concerning Bob's one-time prekeys, if available, the server should provide one, which should be be promptly deleted after use. In the event that all of Bob's one-time prekeys on the server have been deleted, the bundle will not include a one-time prekey.

Step 4a. If the bundle does not contain a one-time prekey, she calculates:

- DH_1 = DH(IK_A, SPK_B)
- DH_2 = DH(EK_A, IK_B)
- DH_3 = DH(EK_A, SPK_B)
- SK = KDF(DH_1 || DH_2 || DH_3)

Step 4b. If the bundle does contain an OPK_B, then an extra DH is performed:

- DH_1 = DH(IK_A, SPK_B)
- DH_2 = DH(EK_A, IK_B)
- DH_3 = DH(EK_A, SPK_B)
- DH_4 = DH(EK_A, OPK_B)
- SK = KDF(DH_1 || DH_2 || DH_3 || DH_4))

The diagram below illustrates the DH calculations between keys. Notably, DH1 and DH2 ensure mutual authentication, while DH3 and DH4 contribute to forward secrecy. For further details on the process and its background, please consult Signal's documentation [link].

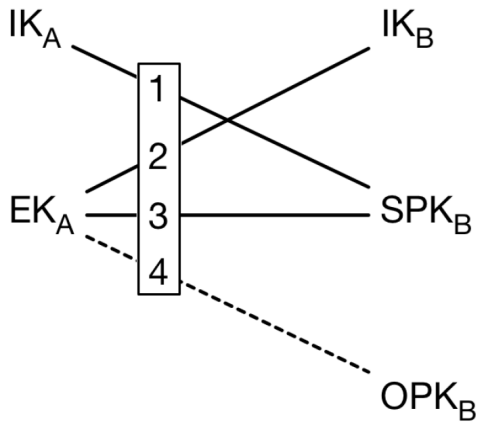


Figure 2.4: DH calculations diagram [2].

Step 5. After computing SK, Alice deletes EK_A and all DH outputs.

Step 6. Alice computes an associated data byte sequence AD that includes identity information for both parties:

AD = Encode(IK_A || Encode(IK_B)

Optionally, Alice can append additional identifying information in AD, such as usernames, certificates, or other relevant details.

Step 7. Alice then transmits an initial message to Bob, which includes:

- Alice's identity key IK_A.
- Alice's ephemeral key EK_A.
- Identifiers indicating the specific prekeys from Bob that Alice used.
- An initial ciphertext encrypted using an AEAD scheme.
  It uses AD as associated data and an encryption key which is either SK or the output from some cryptographic PseudoRandom Function[4] (PRF) keyed by SK.

[4] A PRF generates random-like output from an input and secret key, expanding limited key material into larger pseudorandom data while being deterministic and computationally indistinguishable from true randomness.

The initial ciphertext plays a dual role in the post-X3DH communication protocol. It serves as both the first message within the protocol and as part of Alice's X3DH initial message. Once sent, Alice can utilize SK or keys derived from SK for further communication with Bob within the post-X3DH protocol, taking into account the security considerations outlined in [2].

· **Phase 3 (Receiving the initial message - Alice)**

Finally, for the third phase of the protocol, Bob does the following:

Step 1. Bob retrieves Alice's IK_A and EK_A from the initial message.
Step 2. Bob loads the corresponding private keys for IK_B, SPK_B and OPK_B (if a one-time prekey was used by Alice).
Step 3. Bob repeats DH and KDF calculations to derive SK using the retrieved keys.
Step 4. After computing SK, Bob deletes all DH values.
Step 5. Bob constructs the AD byte sequence using IK_A and IK_B, as in Phase 2.
Step 6. Bob attempts to decrypt the initial ciphertext using SK and AD.
- If decryption fails, abort the protocol and delete SK.
- If decrypts succeeds, the protocol is complete for Bob.

Step 7. Delete any one-time prekey private key that was used, for forward secrecy.

After the protocol has been completed Bob can continue using SK or keys derived from SK within the post-X3DH protocol for communication with Alice, following the security considerations in [2]. For more information on these, as well as for performance considerations, the interested reader is referred to [2].

Remark. Parties engaging in an X3DH key agreement have the option to compare their identity public keys either before or after the agreement. This comparison can be done through an authenticated channel. For instance, they can manually compare public key fingerprints or use QR codes for comparison.

It is important to note that without authentication, parties communicating with each other lack cryptographic assurance of each other's identity.

2.3 Double Ratchet Algorithm

The Double Ratchet algorithm facilitates encrypted message exchange between two parties who share a secret key. Initially, a key agreement protocol like X3DH is used to establish the shared secret key. Once established, the Double Ratchet is employed to securely send and receive encrypted messages. To ensure forward secrecy, new keys are derived for each Double Ratchet message, preventing the calculation of earlier keys from later ones. Diffie-Hellman public values are included with the messages and mixed into the derived keys, further safeguarding future party's keys. Hence, this approach offers protection to both earlier and later encrypted messages.

The Double Ratchet, along with its header encryption variant and the security properties for both are thoroughly discussed in [3]. For our specific needs, we will discuss the inner workings of the Double Ratchet algorithm without the header. More detailed information can be found in [3] for those seeking further understanding.

Lastly, we note that certain diagrams in this section take inspiration from [3], but have been creatively expanded upon to include more intricate details. These diagrams have been merged into a controllable GIF (that might not work when viewed from a phone) to present a more vivid representation of the inner workings of the Double Ratchet algorithm.

2.3.1 KDF Chains

In the context of the Double Ratchet algorithm, a KDF plays a crucial role.

Definition 2.3. A KDF chain is defined as a cryptographic function that takes a secret and random KDF key, along with input data, and returns output data, where:

1. The output data should be indistinguishable from random, if the key remains unknown.
2. The KDF should provide a secure cryptographic hash of its key and input data, regardless of the key's secrecy or randomness.

Notably, constructions like HMAC (Hash-based Message Authentication Code) and HKDF, instantiated with a secure hash algorithm, fulfill the requirements of a KDF [23, 28].

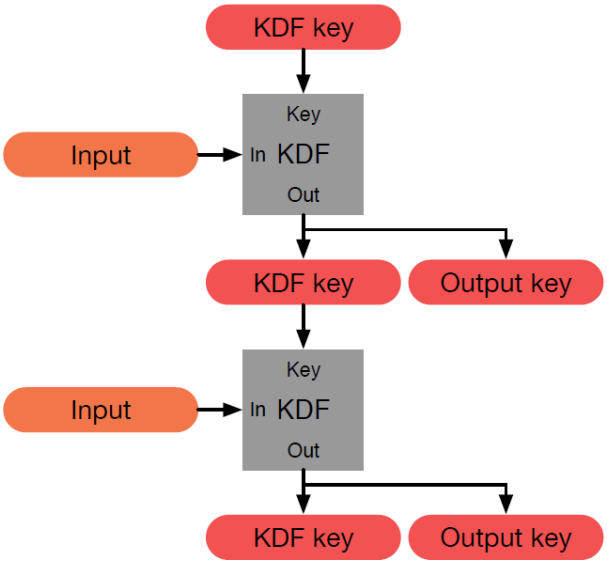


Figure 2.5: KDF chain processing two inputs and producing two output keys [3].

Within the Double Ratchet algorithm, the term "KDF chain" is used to describe the process where a portion of the output from a KDF is employed as an output key, while the remaining portion replaces the KDF key itself. This replaced key can then be used in conjunction with another input, facilitating the generation of subsequent keys (see figure 2.5).

Furthermore, a KDF chain can manifest the following properties:

Property	Description
Resilience	The output keys remain indistinguishable from random to an adversary that doesn't have the KDF keys. This remains true even when the adversary can control the KDF inputs.
Forward Security	Output keys from the past appear random to an adversary who learns the KDF key at a later point in time.
Break-in Recovery	Future output keys appear random to an adversary who learns the KDF key, as long as future inputs introduce enough entropy.

Table 2.3: Properties of a KDF Chain

· **KDF Chains and Ratchets**

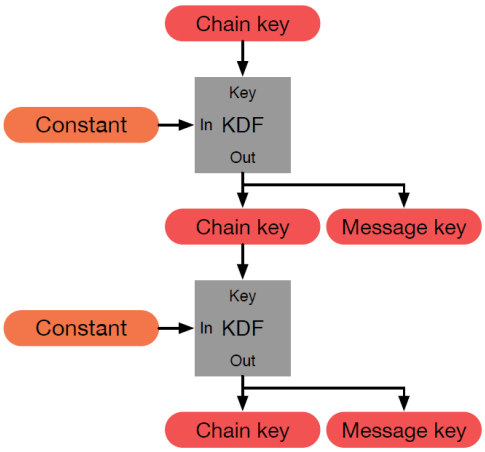
In a Double Ratchet session, Alice and Bob each store KDF keys for three chains: a root chain, a sending chain, and a receiving chain. The sending chain of Alice corresponds to the receiving chain of Bob, and vice versa.

During message exchange, Alice and Bob update their Diffie-Hellman public keys and utilize the resulting secrets as inputs to the root chain. The output keys from the root chain then serve as new KDF keys for the sending and receiving chains. This process, is known as the Diffie-Hellman ratchet[5]. With every message sent and received, the sending and receiving chains progress forward. The output keys derived from these chains are used for symmetric-key encryption and decryption, constituting the Symmetric-key ratchet.

In general, the sending and receiving chains have forward secrecy but do not offer any protection for past messages, as their input is constant. To adress this, the Diffie-Hellman ratchet was introduced, which will be elaborated upon in a following segment.

2.3.2 Symmetric-key Ratchet

Each message is encrypted/ decrypted using a distinct message key, which is an output key from the sending or receiving KDF chain, respectively. The KDF keys for these chains will be called chain keys (see figure, taken from [3]).



Moreover, since the KDF inputs for the sending and receiving chains remain constant, they do not offer break-in recovery. Instead, their purpose is to guarantee that each message is encrypted / decrypted with a unique key, which can be discarded after encryption or decryption. The process of generating the next chain key and message key from a given chain key is referred to as a single ratchet step in the symmetric-key ratchet.

Finally, we specify that message keys in the Double Ratchet algorithm can be stored independently without impacting the security of other message keys. This characteristic proves useful when managing lost or out-of-order messages, as we will see in 2.3.5.

[5]In general, a ratchet is a mechanism that allows movement in one direction while preventing motion in the opposite direction.

2.3.3 Diffie-Hellman Ratchet

As we mentioned before, if an attacker gains access to both the sending and receiving chain keys of one party, they can calculate all future message keys and decrypt forthcoming messages. To mitigate this threat, the Double Ratchet algorithm combines the symmetric-key ratchet with a Diffie-Hellman (DH) ratchet, which updates the chain keys based on Diffie-Hellman outputs.

To implement the DH ratchet, each party generates a Diffie-Hellman key pair consisting of a public key and a private key. The current ratchet key pair is determined by the party's DH key pair. Each message includes a header containing the sender's current ratchet public key. When a new ratchet public key is received from the other party, a DH ratchet step is performed, replacing the local party's current ratchet key pair with a new one.

This process creates a "ping-pong" effect as the parties take turns replacing their ratchet key pairs. If an eavesdropper compromises one party temporarily, they may gain knowledge of a current ratchet private key. However, that compromised key will eventually be replaced with an uncompromised one. Consequently, the Diffie-Hellman computation involving the ratchet key pairs will yield an unknown DH output to the attacker.

The following diagrams depict DH ratchet's derivation of a shared sequence of DH outputs:

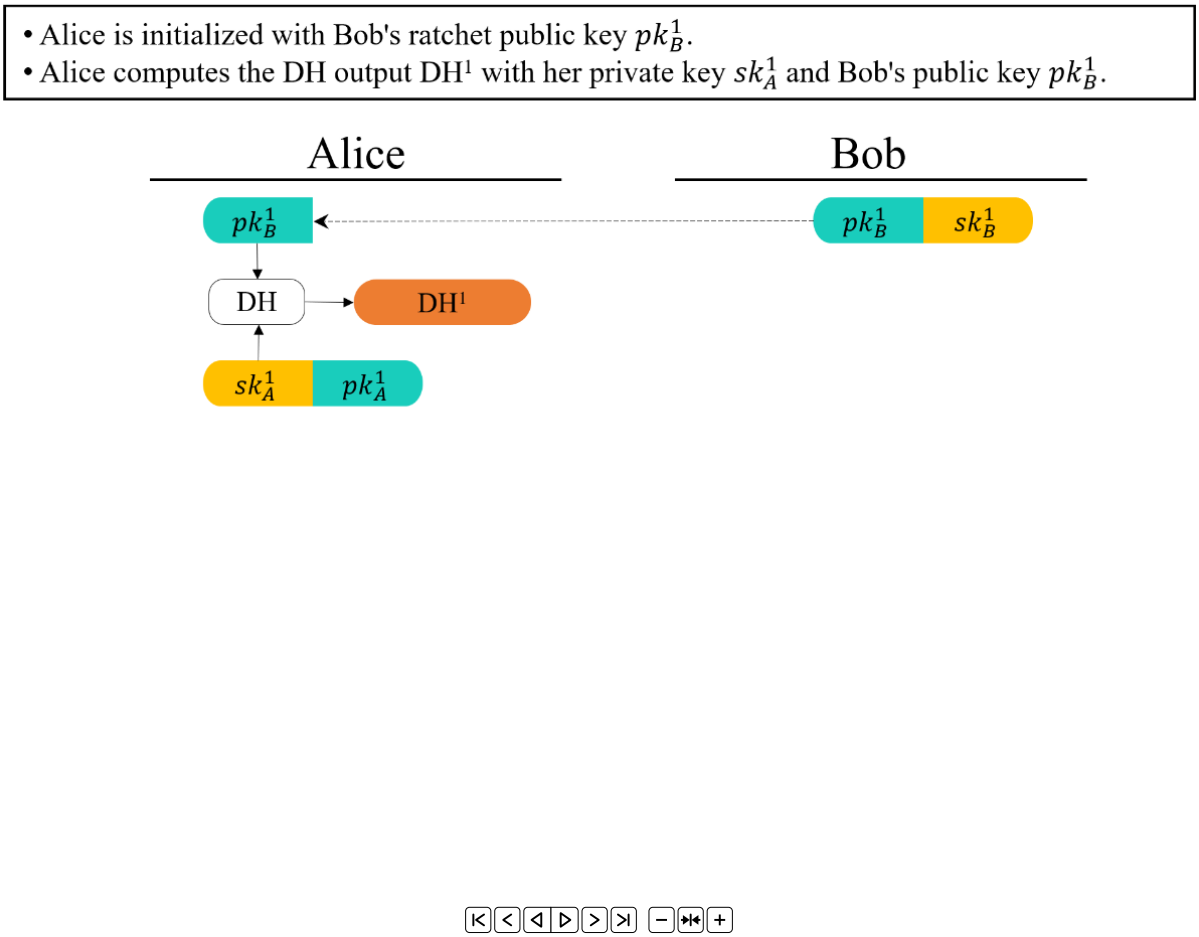


Figure 2.6: Example of Diffie-Hellman ratchet computations. (remake from [3])

2.3.4 Double Ratchet

The Double Ratchet algorithm is achieved by combining the symmetric-key and DH ratchets in the following manner:

- For each message sent or received, a symmetric-key ratchet step is performed on the corresponding sending or receiving chain to derive the message key.
- When a new ratchet public key is received, a DH ratchet step is executed before the symmetric-key ratchet, replacing the chain keys.

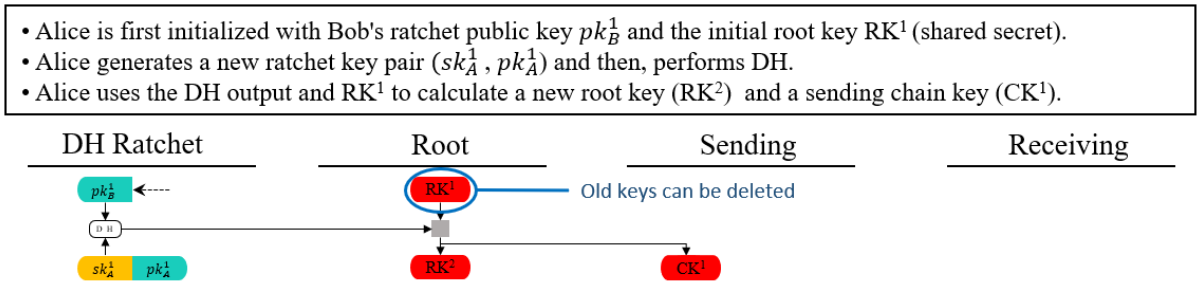


Figure 2.7: Example of Double Ratchet computations. (remake from [3])

2.3.5 Out-of-order messages

The Double Ratchet algorithm effectively handles lost or out-of-order messages by incorporating relevant information in each message header. This information includes the message's number in the sending chain ($N = 0, 1, 2, ...$) and the length (number of message keys) in the previous sending chain (PN). By including this data, the recipient can advance to the appropriate message key while preserving any skipped message keys in anticipation of their potential arrival.

Upon receiving a message, if a DH ratchet step is triggered, the difference between the received PN and the length of the current receiving chain represents the number of skipped

messages within that particular receiving chain. Conversely, the received N denotes the number of skipped messages in the new receiving chain following the DH ratchet step.

In cases where a DH ratchet step is not triggered, the difference between the received N and the length of the receiving chain signifies the number of skipped messages within that chain.

Case	**Total Skipped Messages (Current + New Chain)**
DH ratchet step triggered	$(PN - \text{length of current receiving chain}) + N$
DH ratchet step not triggered	$(N - \text{length of receiving chain}) + 0$

To illustrate, let's consider the message sequence mentioned in the previous section where messages $B2$ and $B3$ were skipped. When Alice receives message $B4$, it triggers her DH ratchet step (instead of $B3$). At this point, $B4$ will have $PN = 2$ and $N = 1$ and therefore the total skipped keys will be

$$(PN - \text{length of current receiving chain}) + N = 2 - 1 + 1 = 2$$

. Hence, since Alice's receiving chain only consists of one message key ($B1$), she will store message keys for $B2$ and $B3$ in anticipation of their potential arrival and decryption.

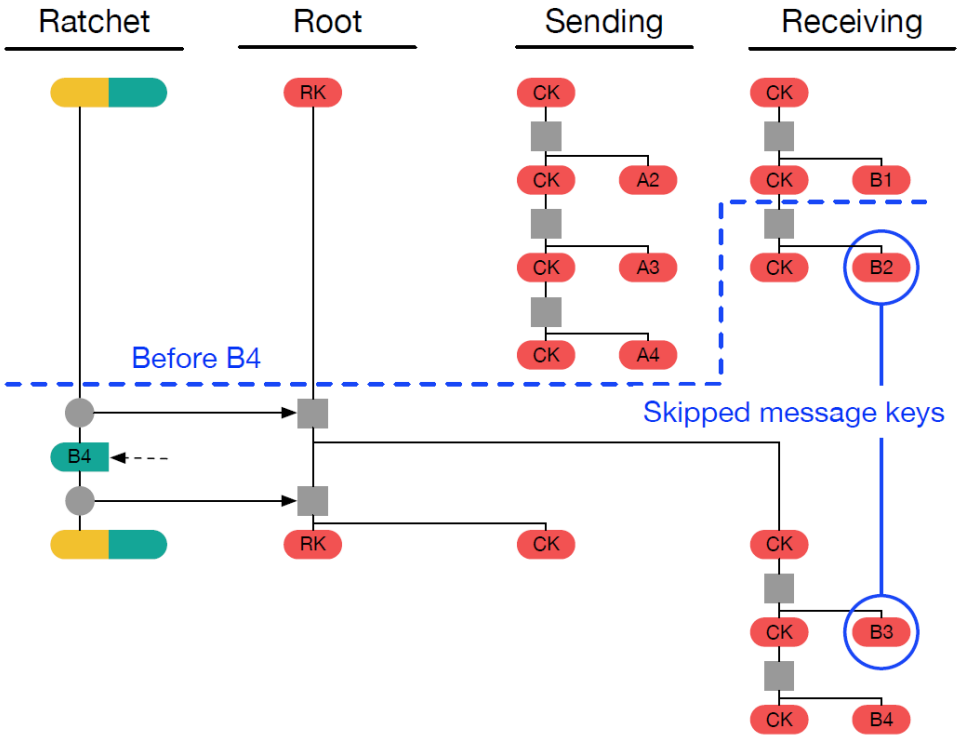


Figure 2.8: Example - Double Ratchet's approach to handling lost or out-of-order messages [3].

2.4 Conclusion and future prospects

In conclusion, X3DH and Double Ratchet, as well as their implementation inside Signal Protocol, represent remarkable advancements in secure communication protocols and encryption methods. These technologies have revolutionized the way we communicate by ensuring end-to-end encryption, strong security guarantees, and protection against unauthorized access.

The Signal Protocol, developed by Open Whisper Systems, has become the gold standard for secure messaging applications. Its robust encryption algorithms and forward secrecy mechanisms provide users with the confidence that their messages and data remain private and secure.

X3DH, an essential component of the Signal Protocol, enables secure key exchange between two parties over an insecure channel. By combining asymmetric and symmetric encryption techniques, X3DH establishes a shared secret key, ensuring that only the intended recipients can decrypt the messages.

The Double Ratchet algorithm, integrated within the Signal Protocol, enhances security further by providing forward secrecy and message integrity. It employs innovative techniques such as symmetric encryption, key derivation, and ratcheting mechanisms to ensure that even if encryption keys are compromised, past and future messages remain protected.

Together, the Signal Protocol, X3DH, and Double Ratchet offer a comprehensive framework for secure and private communication. These technologies have gained widespread adoption, with many messaging apps implementing them to safeguard user information and maintain confidentiality.

Post-quantum Signal Protocol

The Signal Protocol, while offering strong security features, relies on the ECDH key exchange, which is vulnerable to attacks from quantum computers. To address this, a post-quantum Signal Protocol is needed, employing quantum-resistant cryptographic algorithms to ensure secure communication in a quantum-enabled world.

PQShield's research paper proposes a post-quantum upgrade to the Signal protocol, addressing the quantum threat to secure messaging. The integration of post-quantum cryptography into the Signal protocol is essential to ensure continued security in the face of quantum attacks. For more on this topic, the interested reader is referred to PQShield's whitepapers [link].

References

[1] Trevor Perrin. *The XEdDSA and VXEdDSA Signature Schemes*. 2016. URL: https://signal.org/docs/specifications/xeddsa/.

[2] Moxie Marlinspik and Trevor Perrin. *The X3DH Key Agreement Protocol*. 2016. URL: https://signal.org/docs/specifications/x3dh/.

[3] Moxie Marlinspik and Trevor Perrin. *The Double Ratchet Algorithm*. 2016. URL: https://signal.org/docs/specifications/doubleratchet/.

[4] Moxie Marlinspik and Trevor Perrin. *The XEdDSA and VXEdDSA Signature Schemes*. 2017. URL: https://signal.org/docs/specifications/sesame/.

[5] L.C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition*. Discrete Mathematics and Its Applications. CRC Press, 2008. ISBN: 9781420071474. URL: https://books.google.gr/books?id=nBfCEqpYKW0C.

[6] Joseph Silverman. *The Arithmetic of Elliptic Curves*. Vol. 106. Jan. 2009. ISBN: 978-0-387-09493-9. DOI: 10.1007/978-0-387-09494-6.

[7] Christophe Ritzenthaler. *Elliptic curves and applications to cryptography (notes)*. 2011-2012.

[8] Harold M. Edwards. "A normal form for elliptic curves". In: *Bulletin of the American Mathematical Society* 44 (2007), pp. 393–422.

[9] Daniel Bernstein and Tanja Lange. "Faster Addition and Doubling on Elliptic Curves". In: vol. 2007. Dec. 2007, p. 286. ISBN: 978-3-540-76899-9.

[10] Daniel J. Bernstein and Tanja Lange. "Montgomery curves and the Montgomery ladder". In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 293.

[11] Craig Costello and Benjamin Smith. "Montgomery curves and their arithmetic". In: *CoRR* abs/1703.01863 (2017). arXiv: 1703.01863. URL: http://arxiv.org/abs/1703.01863.

[12] Daniel Bernstein et al. “Twisted Edwards Curves”. In: vol. 5023. June 2008, pp. 389–405. ISBN: 978-3-540-68159-5. DOI: 10.1007/978-3-540-68164-9_26.

[13] Adam Langley, Mike Hamburg, and Sean Turner. *Elliptic Curves for Security*. RFC 7748. Jan. 2016. DOI: 10.17487/RFC7748. URL: https://www.rfc-editor.org/info/rfc7748.

[14] Daniel Bernstein et al. “High-Speed High-Security Signatures”. In: vol. 2. Sept. 2011, pp. 124–142. ISBN: 978-3-642-23950-2. DOI: 10.1007/978-3-642-23951-9_9.

[15] Daniel J. Bernstein et al. “EdDSA for more curves”. In: *IACR Cryptol. ePrint Arch.* 2015 (2015), p. 677.

[16] Daniel J. Bernstein. “Curve25519: New Diffie-Hellman Speed Records”. In: *International Conference on Theory and Practice of Public Key Cryptography*. 2006.

[17] Michael Hamburg. “Ed448-Goldilocks, a new elliptic curve”. In: *IACR Cryptol. ePrint Arch.* 2015 (2015), p. 625.

[18] Yevgeniy Dodis and Aleksandr Yampolskiy. “A Verifiable Random Function With Short Proofs and Keys.” In: *IACR Cryptology ePrint Archive* 2004 (Jan. 2004), p. 310.

[19] S. Micali, M. Rabin, and S. Vadhan. “Verifiable random functions”. In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*. 1999, pp. 120–130. DOI: 10.1109/SFFCS.1999.814584.

[20] Andrey Jivsov. “Compact representation of an elliptic curve point”. In: Internet Engineering Task Force, 2014. URL: https://www.ietf.org/archive/id/draft-jivsov-ecc-compact-05.txt.

[21] National Institute of Standards and Technology. *Secure Hash Standard (SHS)*. Federal Information Processing Standards Publication FIPS 180-4. U.S. Department of Commerce, 2012. URL: https://csrc.nist.gov/publications/detail/fips/180/4/final.

[22] Moxie Marlinspike. *Simplifying OTR deniability*. 2013. URL: https://signal.org/blog/simplifying-otr-deniability/.

[23] Dr. Hugo Krawczyk and Pasi Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. RFC 5869. May 2010. DOI: 10.17487/RFC5869. URL: https://www.rfc-editor.org/info/rfc5869.

[24] Lidong Chen. *SP 800-108. Recommendation for Key Derivation Using Pseudorandom Functions (Revised)*. Tech. rep. Gaithersburg, MD, USA, 2009.

[25] Hugo Krawczyk. “Cryptographic Extraction and Key Derivation: The HKDF Scheme”. In: *IACR Cryptology ePrint Archive*. 2010.

[26] David McGrew. *An Interface and Algorithms for Authenticated Encryption*. RFC 5116. Jan. 2008. DOI: 10.17487/RFC5116. URL: https://www.rfc-editor.org/info/rfc5116.

[27] Armando Ruggeri et al. “BCB-X3DH: a Blockchain Based Improved Version of the Extended Triple Diffie-Hellman Protocol”. In: *2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)* (2020), pp. 73–78.

[28] Dr. Hugo Krawczyk, Mihir Bellare, and Ran Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. Feb. 1997. DOI: 10.17487/RFC2104. URL: https://www.rfc-editor.org/info/rfc2104.