>>> EdDSA: Not just ECDSA with a twist

>>> Math 445 (Introduction to Cryptography)

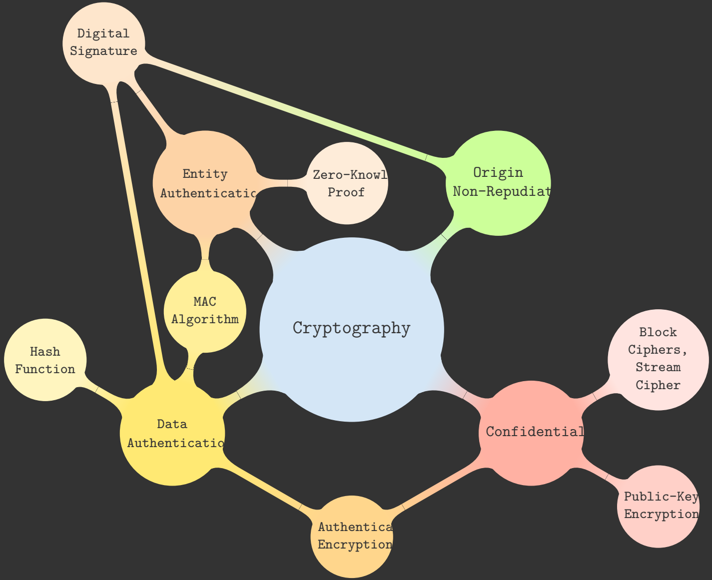
Name: Gaurish Korpal (University of Arizona)

Date: April 29, 2024

\>>> Table of contents

1. Setup

2. Punchline



>>> Setup

>>> HTTPS

1. Visit https://letsencrypt.org/

>>> HTTPS

1. Visit https://letsencrypt.org/
2. View security information in your browser.

>>> NIST

1. https://www.nist.gov/news-events/news/2023/02/
nist-revises-digital-signature-standard-dss-and-publishes-guideline

>>> NIST

1. https://www.nist.gov/news-events/news/2023/02/nist-revises-digital-signature-standard-dss-and-publishes-guideline
2. Use better browsers like Firefox which support ECDSA.

>>> NIST

1. https://www.nist.gov/news-events/news/2023/02/nist-revises-digital-signature-standard-dss-and-publishes-guideline
2. Use better browsers like Firefox which support ECDSA.
   * https://www.keylength.com/en/compare/

>>> ECDSA

Choose a cryptographic hash function H with appropriate domain and codomain. The key generation algorithm outputs a pair (k, Q) such that $Q = [k]P \in \mathbb{G} = \langle P \rangle$ with $|\mathbb{G}| = \ell \nmid p$, where k is the *secret signing key* and Q is the *public verification key*.

>>> ECDSA

Choose a cryptographic hash function H with appropriate domain and codomain. The key generation algorithm outputs a pair (k, Q) such that $Q = [k]P \in \mathbb{G} = \langle P \rangle$ with $|\mathbb{G}| = \ell \nmid p$, where k is the *secret signing key* and Q is the *public verification key*.

Signing $(\mathbb{G}, P, k, \mathsf{H}, m)$

1. $t \xleftarrow{\$} \{1, \ldots, \ell - 1\}$
2. $R \leftarrow [t]P$
3. $r \leftarrow x(R) \pmod{\ell}$
4. if $r = 0$ then goto Step 1.
5. $e \leftarrow \mathsf{H}(m)$
6. $s \leftarrow (e + rk)t^{-1} \pmod{\ell}$
7. if $s = 0$ then goto Step 1.
8. return $\sigma := (r, s)$

>>> ECDSA

Choose a cryptographic hash function H with appropriate domain and codomain. The key generation algorithm outputs a pair (k, Q) such that $Q = [k]P \in \mathbb{G} = \langle P \rangle$ with $|\mathbb{G}| = \ell \nmid p$, where k is the *secret signing key* and Q is the *public verification key*.

Signing $(\mathbb{G}, P, k, \mathsf{H}, m)$

1. $t \xleftarrow{\$} \{1, \ldots, \ell - 1\}$
2. $R \leftarrow [t]P$
3. $r \leftarrow x(R) \pmod{\ell}$
4. if $r = 0$ then goto Step 1.
5. $e \leftarrow \mathsf{H}(m)$
6. $s \leftarrow (e + rk)t^{-1} \pmod{\ell}$
7. if $s = 0$ then goto Step 1.
8. return $\sigma := (r, s)$

Verification $(\mathbb{G}, P, Q, \mathsf{H}, m, \sigma)$

1. $e \leftarrow \mathsf{H}(m)$
2. $u_1 \leftarrow es^{-1} \pmod{\ell}$, $u_2 \leftarrow rs^{-1} \pmod{\ell}$
3. $T \leftarrow [u_1]P + [u_2]Q$
4. return $r \stackrel{?}{=} x(T) \pmod{\ell}$

>>> Warning

1. ECDSA requires a good source of entropy because the ephemeral secret t needs to be truly random.

>>> Warning

1. ECDSA requires a good source of entropy because the ephemeral secret t needs to be truly random.
2. ECDSA was invented only to circumvent patents in Schnorr signatures. Unfortunately, ECDSA does not come with a proof of security, while Schnorr signatures did.

>>> Failures

1. Sony PlayStation 3:
   https://en.wikipedia.org/wiki/PlayStation_3_homebrew

>>> Failures

1. Sony PlayStation 3:
   https://en.wikipedia.org/wiki/PlayStation_3_homebrew
   * A failure to choose random ephemeral values allowed attackers to determine the private key for signing all applications.

>>> Failures

1. Sony PlayStation 3: https://en.wikipedia.org/wiki/PlayStation_3_homebrew
   * A failure to choose random ephemeral values allowed attackers to determine the private key for signing all applications.
2. NSA: https://en.wikipedia.org/wiki/Dual_EC_DRBG

>>> Failures

1. Sony PlayStation 3: https://en.wikipedia.org/wiki/PlayStation_3_homebrew
   * A failure to choose random ephemeral values allowed attackers to determine the private key for signing all applications.
2. NSA: https://en.wikipedia.org/wiki/Dual_EC_DRBG
   * At the CRYPTO 2007 conference rump session, Dan Shumow and Niels Ferguson presented a potential backdoor in the NIST/NSA specified Dual_EC_DRBG cryptographically secure pseudorandom number generator. The backdoor was confirmed to be real in 2013 as part of the Edward Snowden leaks.

>>> Punchline

>>> Elliptic curves

Let p be a prime larger than 3 and $q = p^n$ for $n > 0$. Elliptic curves can be represented with several different types of defining equations over $\mathbb{F}_q$.

>>> Elliptic curves

Let p be a prime larger than 3 and $q = p^n$ for $n > 0$. Elliptic curves can be represented with several different types of defining equations over $\mathbb{F}_q$.

1. **Short Weierstrass form:** $y^2 = x^3 + ax + b$ such that $4a^3 + 27b^2 \neq 0$

>>> Elliptic curves

Let p be a prime larger than 3 and $q = p^n$ for $n > 0$. Elliptic curves can be represented with several different types of defining equations over $\mathbb{F}_q$.

1. Short Weierstrass form: $y^2 = x^3 + ax + b$ such that $4a^3 + 27b^2 \neq 0$
    * Every elliptic curve can be represented in this form.

>>> Elliptic curves

Let p be a prime larger than 3 and $q = p^n$ for $n > 0$. Elliptic curves can be represented with several different types of defining equations over $\mathbb{F}_q$.

1. **Short Weierstrass form:** $y^2 = x^3 + ax + b$ such that $4a^3 + 27b^2 \neq 0$
   * Every elliptic curve can be represented in this form.
2. **Montgomery form:** $by^2 = x^3 + ax^2 + x$ such that $b(a^2 - 4) \neq 0$

>>> Elliptic curves

Let p be a prime larger than 3 and $q = p^n$ for $n > 0$. Elliptic curves can be represented with several different types of defining equations over $\mathbb{F}_q$.

1. Short Weierstrass form: $y^2 = x^3 + ax + b$ such that $4a^3 + 27b^2 \neq 0$
   * Every elliptic curve can be represented in this form.
2. Montgomery form: $by^2 = x^3 + ax^2 + x$ such that $b(a^2 - 4) \neq 0$
   * The group order is divisible by 4.

>>> Elliptic curves

Let p be a prime larger than 3 and $q = p^n$ for $n > 0$. Elliptic curves can be represented with several different types of defining equations over $\mathbb{F}_q$.

1. Short Weierstrass form: $y^2 = x^3 + ax + b$ such that $4a^3 + 27b^2 \neq 0$
    * Every elliptic curve can be represented in this form.
2. Montgomery form: $by^2 = x^3 + ax^2 + x$ such that $b(a^2 - 4) \neq 0$
    * The group order is divisible by 4.
    * x-only "differential addition," i.e. $x(P + Q)$ can be computed using only $x(P)$, $x(Q)$, and $x(Q - P)$.

>>> Elliptic curves

Let p be a prime larger than 3 and $q = p^n$ for $n > 0$. Elliptic curves can be represented with several different types of defining equations over $\mathbb{F}_q$.

1. Short Weierstrass form: $y^2 = x^3 + ax + b$ such that $4a^3 + 27b^2 \neq 0$
   * Every elliptic curve can be represented in this form.
2. Montgomery form: $by^2 = x^3 + ax^2 + x$ such that $b(a^2 - 4) \neq 0$
   * The group order is divisible by 4.
   * x-only "differential addition," i.e. $x(P + Q)$ can be computed using only $x(P)$, $x(Q)$, and $x(Q - P)$.
3. Twisted Edwards form: $ax^2 + y^2 = 1 + dx^2y^2$ such that $ad(a - d) \neq 0$

>>> Elliptic curves

Let p be a prime larger than 3 and $q = p^n$ for $n > 0$. Elliptic curves can be represented with several different types of defining equations over $\mathbb{F}_q$.

1. **Short Weierstrass form:** $y^2 = x^3 + ax + b$ such that $4a^3 + 27b^2 \neq 0$
   * Every elliptic curve can be represented in this form.
2. **Montgomery form:** $by^2 = x^3 + ax^2 + x$ such that $b(a^2 - 4) \neq 0$
   * The group order is divisible by 4.
   * x-only "differential addition," i.e. $x(P+Q)$ can be computed using only $x(P)$, $x(Q)$, and $x(Q-P)$.
3. **Twisted Edwards form:** $ax^2 + y^2 = 1 + dx^2y^2$ such that $ad(a-d) \neq 0$
   * Birationally equivalent to Montgomery form.

>>> Elliptic curves

Let p be a prime larger than 3 and $q = p^n$ for $n > 0$. Elliptic curves can be represented with several different types of defining equations over $\mathbb{F}_q$.

1. Short Weierstrass form: $y^2 = x^3 + ax + b$ such that $4a^3 + 27b^2 \neq 0$
   * Every elliptic curve can be represented in this form.
2. Montgomery form: $by^2 = x^3 + ax^2 + x$ such that $b(a^2 - 4) \neq 0$
   * The group order is divisible by 4.
   * x-only "differential addition," i.e. $x(P + Q)$ can be computed using only $x(P)$, $x(Q)$, and $x(Q - P)$.
3. Twisted Edwards form: $ax^2 + y^2 = 1 + dx^2y^2$ such that $ad(a - d) \neq 0$
   * Birationally equivalent to Montgomery form.
   * If a is a square and d is a non-square, then a single addition formula works for all possible inputs.

>>> Safe curves

The Dual_EC_DRBG algorithm was based on P-256 curve

$$y^2 = x^3 - 3x +$$

$$41058363725152142129326129780047268409114441015993725554835256314039467401291$$

modulo $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

>>> Safe curves

The Dual_EC_DRBG algorithm was based on P-256 curve

$$y^2 = x^3 - 3x +$$

$$41058363725152142129326129780047268409114441015993725554835256314039467401291$$

modulo $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

This led to concerns about security of P-256 itself and the following curve emerged as the de facto alternative (https://safecurves.cr.yp.to).

>>> Safe curves

The Dual_EC_DRBG algorithm was based on P-256 curve

$$y^2 = x^3 - 3x +$$

$$41058363725152142129326129780047268409114441015993725554835256314039467401291$$

modulo $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

This led to concerns about security of P-256 itself and the following curve emerged as the de facto alternative (https://safecurves.cr.yp.to).

* **Curve25519:** $y^2 = x^3 + 486662x^2 + x$ modulo $p = 2^{255} - 19$.

>>> Safe curves

The Dual_EC_DRBG algorithm was based on P-256 curve

$$y^2 = x^3 - 3x + 41058363725152142129326129780047268409114441015993725554835256314039467401291$$

modulo $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

This led to concerns about security of P-256 itself and the following curve emerged as the de facto alternative (https://safecurves.cr.yp.to).

* **Curve25519:** $y^2 = x^3 + 486662x^2 + x$ modulo $p = 2^{255} - 19$.
  * This Montgomery curve is used in ECDSA with co-factor 8 subgroup $\mathbb{G} = \langle P \rangle$ such that $x(P) = 9$.

>>> Safe curves

The Dual_EC_DRBG algorithm was based on P-256 curve

$$y^2 = x^3 - 3x +$$

$$41058363725152142129326129780047268409114441015993725554835256314039467401291$$

modulo $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

This led to concerns about security of P-256 itself and the following curve emerged as the de facto alternative (https://safecurves.cr.yp.to).

* Curve25519: $y^2 = x^3 + 486662x^2 + x$ modulo $p = 2^{255} - 19$.
  * This Montgomery curve is used in ECDSA with co-factor 8 subgroup $\mathbb{G} = \langle P \rangle$ such that $x(P) = 9$.
* Ed25519: $-x^2 + y^2 = 1 - \dfrac{121665}{121666}x^2y^2$ modulo $p = 2^{255} - 19$.

>>> Safe curves

The Dual_EC_DRBG algorithm was based on P-256 curve

$$y^2 = x^3 - 3x + 41058363725152142129326129780047268409114441015993725554835256314039467401291$$

modulo $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.
This led to concerns about security of P-256 itself and the following curve emerged as the de facto alternative (https://safecurves.cr.yp.to).

* **Curve25519:** $y^2 = x^3 + 486662x^2 + x$ modulo $p = 2^{255} - 19$.
  * This Montgomery curve is used in ECDSA with co-factor 8 subgroup $\mathbb{G} = \langle P \rangle$ such that $x(P) = 9$.
* **Ed25519:** $-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2$ modulo $p = 2^{255} - 19$.
  * This twisted Edwards curve is birationally equivalent to Curve25519.

>>> EdDSA, Step 1: Σ-protocol

Let $\mathbb{G} = \langle P \rangle$ with $|\mathbb{G}| = \ell \nmid p$.

>>> EdDSA, Step 1: Σ-protocol

Let $\mathbb{G} = \langle P \rangle$ with $|\mathbb{G}| = \ell \nmid p$. The prover $\mathcal{P}$ randomly chooses (secret) $k \in \{0, \ldots, \ell - 1\}$ and publishes $Q = [k]P$.

>>> EdDSA, Step 1: Σ-protocol

Let $\mathbb{G} = \langle P \rangle$ with $|\mathbb{G}| = \ell \nmid p$. The prover $\mathcal{P}$ randomly chooses (secret) $k \in \{0, \dots, \ell - 1\}$ and publishes $Q = [k]P$. Now, $\mathcal{P}$ can prove knowledge of a discrete logarithm k to a verifier $\mathcal{V}$:

$\mathcal{P}$ ---- $\mathbb{G}, p, Q$ ---- $\mathcal{V}$

$$t \xleftarrow{\$} \{0, \dots, \ell - 1\}$$
$$R \leftarrow [t]P$$

$\xrightarrow{\text{commitment: } R}$

$$e \leftarrow \{0, \dots, \ell - 1\}$$

$$s \leftarrow t + ek \pmod{\ell}$$

$$[s]P \stackrel{?}{=} R + [e]Q$$

>>> EdDSA, Step 1: Σ-protocol

Let $\mathbb{G} = \langle P \rangle$ with $|\mathbb{G}| = \ell \nmid p$. The prover $\mathcal{P}$ randomly chooses (secret) $k \in \{0, \ldots, \ell - 1\}$ and publishes $Q = [k]P$. Now, $\mathcal{P}$ can prove knowledge of a discrete logarithm k to a verifier $\mathcal{V}$:

$$\mathcal{P} \overset{\mathbb{G}, p, Q}{\text{-----------}} \mathcal{V}$$

$$t \xleftarrow{\$} \{0, \ldots, \ell - 1\}$$

$$R \leftarrow [t]P$$

$$\xrightarrow{\texttt{commitment: } R}$$

$$e \leftarrow \{0, \ldots, \ell - 1\}$$

$$\xleftarrow{\texttt{challenge: } e}$$

$$s \leftarrow t + ek \pmod{\ell} \qquad [s]P \stackrel{?}{=} R + [e]Q$$

>>> EdDSA, Step 1: Σ-protocol

Let $\mathbb{G} = \langle P \rangle$ with $|\mathbb{G}| = \ell \nmid p$. The prover $\mathcal{P}$ randomly chooses (secret) $k \in \{0, \ldots, \ell - 1\}$ and publishes $Q = [k]P$. Now, $\mathcal{P}$ can prove knowledge of a discrete logarithm k to a verifier $\mathcal{V}$:

$$\mathcal{P} \overset{\mathbb{G}, p, Q}{\text{-----------------------}} \mathcal{V}$$

$$t \overset{\$}{\leftarrow} \{0, \ldots, \ell - 1\}$$

$$R \leftarrow [t]P$$

$$\xrightarrow{\texttt{commitment:}\ R}$$

$$e \leftarrow \{0, \ldots, \ell - 1\}$$

$$\xleftarrow{\texttt{challenge:}\ e}$$

$$s \leftarrow t + ek \pmod{\ell} \xrightarrow{\texttt{response:}\ s} [s]P \overset{?}{=} R + [e]Q$$

>>> EdDSA, Step 2: Fiat-Shamir transformation

Choose a *random oracle* cryptographic hash function H with appropriate domain and codomain.

>>> EdDSA, Step 2: Fiat-Shamir transformation

Choose a *random oracle* cryptographic hash function H with appropriate domain and codomain. The key generation algorithm outputs a pair (k, Q) such that $Q = [k]P \in \mathbb{G} = \langle P \rangle$ with $|\mathbb{G}| = \ell \nmid p$, where k is the *secret signing key* and Q is the *public verification key*.

>>> EdDSA, Step 2: Fiat-Shamir transformation

Choose a *random oracle* cryptographic hash function H with appropriate domain and codomain. The key generation algorithm outputs a pair (k, Q) such that $Q = [k]P \in \mathbb{G} = \langle P \rangle$ with $|\mathbb{G}| = \ell \nmid p$, where k is the *secret signing key* and Q is the *public verification key*.

Signing $(\mathbb{G}, P, k, \mathsf{H}, m)$

1. $t \xleftarrow{\$} \{1, \ldots, \ell - 1\}$
2. $R \leftarrow [t]P$
3. $e \leftarrow \mathsf{H}(m \| R)$
4. $s \leftarrow t + ek \pmod{\ell}$
5. return $\sigma := (R, s)$

>>> EdDSA, Step 2: Fiat-Shamir transformation

Choose a *random oracle* cryptographic hash function H with appropriate domain and codomain. The key generation algorithm outputs a pair (k, Q) such that $Q = [k]P \in \mathbb{G} = \langle P \rangle$ with $|\mathbb{G}| = \ell \nmid p$, where k is the *secret signing key* and Q is the *public verification key*.

Signing $(\mathbb{G}, P, k, \mathsf{H}, m)$

1. $t \xleftarrow{\$} \{1, \ldots, \ell - 1\}$
2. $R \leftarrow [t]P$
3. $e \leftarrow \mathsf{H}(m \| R)$
4. $s \leftarrow t + ek \pmod{\ell}$
5. return $\sigma := (R, s)$

Verification $(\mathbb{G}, P, Q, \mathsf{H}, m, \sigma)$

1. $e \leftarrow \mathsf{H}(m \| R)$
2. return $[s]P \stackrel{?}{=} R + [e]Q$

>>> Demonstration: Ed25519

1. Download binary https://jedisct1.github.io/minisign/

>>> Demonstration: Ed25519

1. Download binary https://jedisct1.github.io/minisign/
2. Generate key pair.

>>> Demonstration: Ed25519

1. Download binary https://jedisct1.github.io/minisign/
2. Generate key pair.
   * minisign.pub contains a concatination of Ed (2 bytes), random bits (8 bytes), and 256-bit public key (32 bytes) stored in base64 format.

>>> Demonstration: Ed25519

1. Download binary https://jedisct1.github.io/minisign/
2. Generate key pair.
   * minisign.pub contains a concatination of Ed (2 bytes), random bits (8 bytes), and 256-bit public key (32 bytes) stored in base64 format.
   * minisign.key contains the encrypted private key derived using scrypt, a password-based key derivation function, and stored in base64 format.

>>> Demonstration: Ed25519

1. Download binary https://jedisct1.github.io/minisign/
2. Generate key pair.
   * minisign.pub contains a concatination of Ed (2 bytes), random bits (8 bytes), and 256-bit public key (32 bytes) stored in base64 format.
   * minisign.key contains the encrypted private key derived using scrypt, a password-based key derivation function, and stored in base64 format.
3. Create and sign a file myfile.txt.

>>> Demonstration: Ed25519

1. Download binary https://jedisct1.github.io/minisign/
2. Generate key pair.
   * minisign.pub contains a concatination of Ed (2 bytes), random bits (8 bytes), and 256-bit public key (32 bytes) stored in base64 format.
   * minisign.key contains the encrypted private key derived using scrypt, a password-based key derivation function, and stored in base64 format.
3. Create and sign a file myfile.txt.
   * myfile.txt.minisig contains the signature for BLAKE2b hash of the file contents, stored in base64 format.

>>> Demonstration: Ed25519

1. Download binary https://jedisct1.github.io/minisign/
2. Generate key pair.
   * minisign.pub contains a concatination of Ed (2 bytes), random bits (8 bytes), and 256-bit public key (32 bytes) stored in base64 format.
   * minisign.key contains the encrypted private key derived using scrypt, a password-based key derivation function, and stored in base64 format.
3. Create and sign a file myfile.txt.
   * myfile.txt.minisig contains the signature for BLAKE2b hash of the file contents, stored in base64 format.
4. Verify the signature using public key.

>>> Demonstration: Ed25519

1. Download binary https://jedisct1.github.io/minisign/
2. Generate key pair.
   * minisign.pub contains a concatination of Ed (2 bytes), random bits (8 bytes), and 256-bit public key (32 bytes) stored in base64 format.
   * minisign.key contains the encrypted private key derived using scrypt, a password-based key derivation function, and stored in base64 format.
3. Create and sign a file myfile.txt.
   * myfile.txt.minisig contains the signature for BLAKE2b hash of the file contents, stored in base64 format.
4. Verify the signature using public key.
   * Origin non-repudiation: Protection against falsely denying having performed the action.

\>>> Demonstration: Ed25519

1. Download binary https://jedisct1.github.io/minisign/
2. Generate key pair.
   * minisign.pub contains a concatination of Ed (2 bytes), random bits (8 bytes), and 256-bit public key (32 bytes) stored in base64 format.
   * minisign.key contains the encrypted private key derived using scrypt, a password-based key derivation function, and stored in base64 format.
3. Create and sign a file myfile.txt.
   * myfile.txt.minisig contains the signature for BLAKE2b hash of the file contents, stored in base64 format.
4. Verify the signature using public key.
   * Origin non-repudiation: Protection against falsely denying having performed the action.
   * Entity authentication: Assurance about the identity of the entity interacting with the system.

>>> Demonstration: Ed25519

1. Download binary https://jedisct1.github.io/minisign/
2. Generate key pair.
   * minisign.pub contains a concatination of Ed (2 bytes), random bits (8 bytes), and 256-bit public key (32 bytes) stored in base64 format.
   * minisign.key contains the encrypted private key derived using scrypt, a password-based key derivation function, and stored in base64 format.
3. Create and sign a file myfile.txt.
   * myfile.txt.minisig contains the signature for BLAKE2b hash of the file contents, stored in base64 format.
4. Verify the signature using public key.
   * Origin non-repudiation: Protection against falsely denying having performed the action.
   * Entity authentication: Assurance about the identity of the entity interacting with the system.
   * Data authentication: Assurance of the integrity of data.

>>> Further reading

- Hüseyin Hışıl, 2010, *Elliptic Curves, Group Law, and Efficient Computation.* §1.1 and 2.3.4
  https://eprints.qut.edu.au/33233/
- Nigel P. Smart, 2016, *Cryptography Made Simple.* §21.3
- Simon Josefsson and Ilari Liusvaara, 2017, *Edwards-Curve Digital Signature Algorithm.* §1
  https://www.rfc-editor.org/info/rfc8032
- Steven D. Galbraith, 2018, *Mathematics of Public Key Cryptography.* §9.12
  https://www.math.auckland.ac.nz/~sgal018/crypto-book/crypto-book.html
- David Wong, 2021, *Real World Cryptography.* §7.3.4
- Luís T. A. N. Brandão and Michael Davidson, 2022, *Notes on Threshold EdDSA/Schnorr Signatures.* Figures 1 and 2
  https://csrc.nist.gov/pubs/ir/8214/b/ipd