Implementation of EdDSA and LSB Watermarking for Lossless Audio Authentication

Felix Limanta
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
felixlimanta@gmail.com

***Abstract*—This paper proposes a watermarking scheme used to embed and verify a digital signature for a lossless audio file. The scheme combines EdDSA (Edwards-curve Digital Signature Algorithm) and LSB (Least Significant Bit) watermarking. This paper will discuss the design of the proposed scheme, an implementation as a proof-of-concept, and thorough testing for the three security concepts supported by a digital signature: authentication, integrity, and non-repudiation. Test results show that EdDSA can be implemented and embedded on a lossless audio file without any discernable differences in audio contents and file size.**

Keywords—EdDSA; LSB watermarking; digital audio; digital signature

I. Introduction

As the Internet becomes more easily accessible, information exchange via the Internet. Social interaction in the virtual world becomes more common, using either social media or Internet-based social applications. Such information exchange includes the exchange of digital audio files. Currently, many websites and web applications offer a cloud-based digital audio sharing service; that is, the audio file is not stored in a local hard drive, but in the cloud. Examples of this service include SoundCloud and Spotify.

These services are enjoyed by many people: from listening to professionally-made music to sharing a recording of their own production. Nevertheless, the ease of access of digital audio files is not usually bundled with security: integrity and validity of the audio file, authentication of the audio file owner, or whether the audio file has been tampered or not. Some digital audio sharing services implement complex policies to prevent the abuse of its audio files by irresponsible or malicious parties.

Even so, a digital audio file can be easily disseminated through the Internet and modified by said irresponsible or malicious parties. The contents of the audio file can be modified slightly as to not trip audio copyright violation detection algorithms. In addition, the author of the audio file can easily repudiate its ownership because there is no method to verify which party is the actual owner of the audio.

This paper proposes an audio watermarking scheme, where the necessary information is embedded within the audio files themselves, and such no other files or tags are needed. Therefore, regardless of whether the audio file is downloaded or spread through the Internet, the integrity and ownership of the audio file can still be verified. The proposed solution uses a digital signature with EdDSA combined with LSB steganography to watermark the audio.

II. Theoretical Building Blocks

A. Twisted Edwards Curve

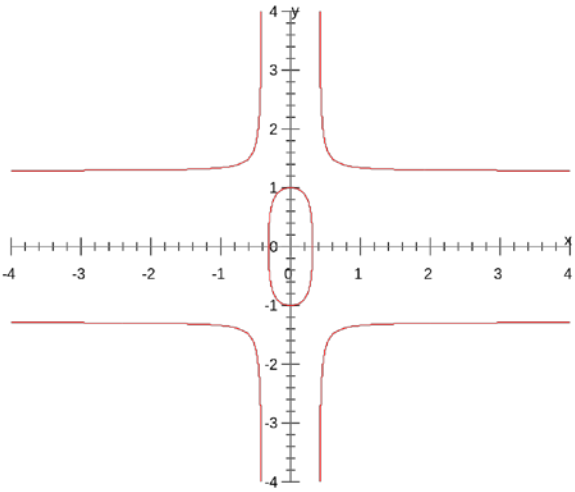

Fig. 1. A twisted Edwards curve of the equation $10x^2 + y^2 = 1 + 6x^2y^2$
Source: Wikimedia Commons

A point $P = (x, y)$ lies on a twisted Edwards curve E if it satisfies the following formula:

$$ax^2 + y^2 = 1 + dx^2y^2 \quad (1)$$

in which a, d are two different non-zero elements of the field $\mathbb{K}$ over which E is defined. For example, edwards25519 is defined over F_p, where $p = 2^{255} - 19$.

A critical property of an elliptic curve, including the twisted Edwards curve, is the presence of the point addition operation ("+"). For the twisted Edwards curve, point addition is defined as:

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2+x_2y_1}{1+dx_1x_2y_1y_2}, \frac{y_1y_2+ax_1x_2}{1-dx_1x_2y_1y_2}\right) \quad (2)$$

The formula can be used both for point addition and point doubling, even with the neutral element $(0, 1)$. Doubling on a twisted Edwards curve is defined as:

$$2 \cdot (x, y) = \left(\frac{2xy}{ax^2y^2}, \frac{y^2-ax^2}{2-ax^2-y^2}\right) \quad (3)$$

Scalar multiplication $n \cdot P$ is defined as adding a point P to itself.

B. *Extended Twisted Edwards Curve*

A point in the twisted Edwards curve can be represented by a different coordinate system other than Cartesian [1]. A point (x, y) can be represented by (X, Y, Z, T) satisfying the following set of equations.

$$x = \frac{\mathrm{X}}{\mathrm{Z}}$$
$$y = \frac{\mathrm{Y}}{\mathrm{Z}} \quad (4)$$
$$x \times y = \frac{\mathrm{T}}{\mathrm{Z}}$$

The neutral point $(0, 1)$ is equivalent to $(0, \mathrm{Z}, \mathrm{Z}, 0)$ in extended homogenous coordinates for any non-zero Z.

Expressing a point in this coordinate system saves time in addition, which can be costly due to the modular division involved. The following algorithm for adding two points on a twisted Edwards curve with $a = -1$ is described in [1].

Algorithm 1 Extended twisted Edwards point addition

Require: $d, (\mathrm{X}_1, \mathrm{Y}_1, \mathrm{Z}_1, \mathrm{T}_1), (\mathrm{X}_2, \mathrm{Y}_2, \mathrm{Z}_2, \mathrm{T}_2)$

1. $A \leftarrow (\mathrm{Y}_1 - \mathrm{X}_1) \times (\mathrm{Y}_2 - \mathrm{X}_2)$
2. $B \leftarrow (Y_1 + X_1) \times (Y_2 + X_2)$
3. $C \leftarrow 2d\mathrm{T}_1\mathrm{T}_2$
4. $D \leftarrow 2\mathrm{Z}_1\mathrm{Z}_2$
5. $E \leftarrow B - A$
6. $F \leftarrow D - C$
7. $G \leftarrow D + C$
8. $H \leftarrow B + A$
9. $\mathrm{X}_3 \leftarrow EF$
10. $\mathrm{Y}_3 \leftarrow GH$
11. $T_3 \leftarrow EH$
12. $Z_3 \leftarrow FG$
13. **return** $(\mathrm{X}_3, \mathrm{Y}_3, \mathrm{Z}_3, \mathrm{T}_3)$

For point doubling, the above algorithm could be used with equal points as input. Nevertheless, another formula, also described in [1], saves a few smaller operations.

Algorithm 2 Extended twisted Edwards point doubling

Require: $(\mathrm{X}_1, \mathrm{Y}_1, \mathrm{Z}_1, \mathrm{T}_1)$

1. $A \leftarrow \mathrm{X}_1^2$
2. $B \leftarrow \mathrm{Y}_1^2$
3. $C \leftarrow 2\mathrm{Z}_1^2$
4. $H \leftarrow B + A$
5. $E \leftarrow H - (\mathrm{X}_1^2 + \mathrm{Y}_1^2)^2$
6. $G \leftarrow A - B$
7. $F \leftarrow C + G$
8. $\mathrm{X}_3 \leftarrow EF$
9. $\mathrm{Y}_3 \leftarrow GH$
10. $T_3 \leftarrow EH$
11. $Z_3 \leftarrow FG$
12. **return** $(\mathrm{X}_3, \mathrm{Y}_3, \mathrm{Z}_3, \mathrm{T}_3)$

C. *Edwards-curve Digital Signature Algorithm (EdDSA)*

EdDSA is a public-key signature algorithm similar to ECDSA proposed by Bernstein et al. in [2]. In RFC 8032 [3], EdDSA is defined for two twisted Edwards curves: edwards25519 and edwards448, but EdDSA can also be used on other curves.

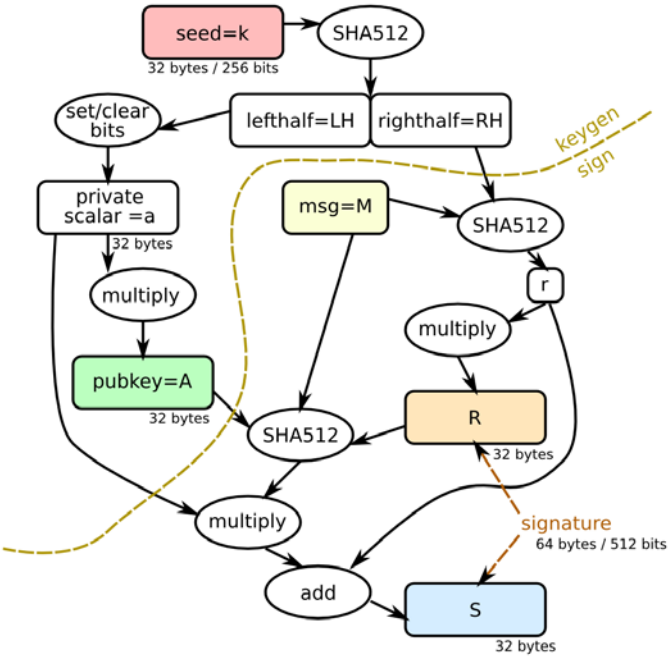

Fig. 2. Illustration of the key generation and signing process of EdDSA
Source: Brian Warner

EdDSA uses a randomly-generated array of bytes with bitlength of b as its signing key k. In addition, EdDSA also requires a hash function H with a $2b$-bits output length. A common occurrence is the usage of SHA512 for $b = 256$ bits. An integer a is calculated from $H(k) = (h_0, h_1, \ldots, h_{2b-1})$ with:

$$a = 2^{b-2} + \sum_{3 \le i \le b-3} 2^i h_i \quad (5)$$

The public key A is computed from a base point $B \neq (0, 1)$ of order ℓ (chosen following the EdDSA specification [2]), such that $A = a \cdot B$.

The signature (R, S) of a message M is generated according to the following algorithm.

Algorithm 3 EdDSA Signature Generation

Require: $M, (h_0, h_1, \dots, h_{2b-1}), B, A$

1. $a \leftarrow 2^{b-2} + \sum_{3 \le i \le b-3} 2^i h_i$
2. $h \leftarrow H(h_b, \dots, h_{2b-1}, M)$
3. $r \leftarrow h \bmod \ell$
4. $R \leftarrow r \cdot B$
5. $h \leftarrow H(R, A, M)$
6. $S \leftarrow (r + ah) \bmod \ell$
7. **return** (R, S)

A signature is considered valid if $R \in E, S \in [0, \ell - 1]$, and the following equation holds:

$$8S \cdot B = 8 \cdot R + 8H(R, A, M) \cdot A. \quad (6)$$

In practice, the public key and the signatures are output according to the encoding specified in RFC 8032 [3]. A point (x, y) is encoded as a *b*-bit string $ENC(x, y)$, which is the *(b-1)*-bit little-endian encoding of *y* concatenated with a parity bit (1 if *x* is positive, 0 otherwise). The value *x* can be recovered with the following equation

$$x_A = \pm\sqrt{(y_A^2 - 1)/(dy_A^2 + 1)} \bmod p \quad (7)$$

Signature computations are deterministic; for a given message *M*, multiple computations of signatures will produce identical results. Like other discrete-log-based signature schemes, EdDSA uses a nonce unique to each signature. In the signature schemes DSA and ECDSA, this nonce is generated randomly for each signature; if the random number generator is predictable when making a signature, the signature can leak the signing key (as happened with the Sony PlayStation 3 firmware update signing key).

In contrast, EdDSA chooses the nonce deterministically as the hash of the signing key and the message. Thus, once a private key is generated, EdDSA no longer has any further need for a random number generator in order to make signatures, and there is no danger that a broken random number generator used to make a signature will reveal the signing key.

D. LSB Watermarking

A digital watermark is a marker covertly embedded in a digital object, such as an audio, video, or image data. It is typically used to identify ownership of the copyright of said data. Watermarking is the process of hiding digital information in a cover data. Digital watermarks may be used to verify the authenticity or integrity of the carrier signal or to show the identity of its owners.

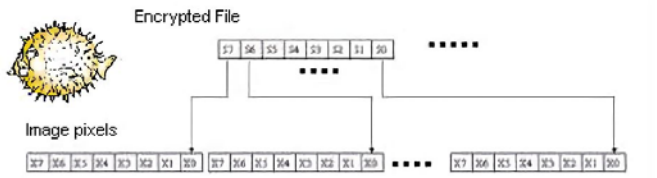

Fig. 3. LSB steganography illustration
Source: codeproject.com

For multimedia data, the most common method of watermark embedding is the LSB substitution, in which the least significant bits of each byte of the cover data are replaced by the embedded data. By only editing the least significant bits of each byte, changes from the original cover data are nearly imperceptible to the human senses.

Despite its simplicity, LSB substitution is considered fragile watermarking. Although transformations like cropping are survivable, any addition of undesirable noise or lossy compression could destroy the embedded message.

E. Raw Audio

Raw audio is the most basic data representation of audio. Raw audio lacks a signature and a header, which makes determining a raw audio's file format significantly harder than interchange formats due to said lack of signature. In addition, the lack of a commonly understood header makes determining audio parameters difficult.

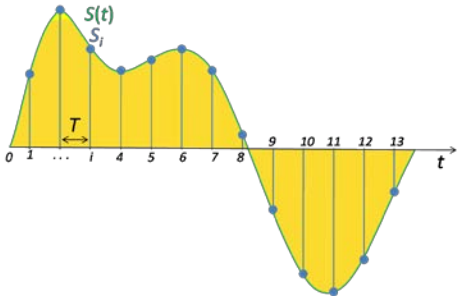

Fig. 4. Signal sampling representation
Source: Wikimedia Commons

Digital audio is stored as samples. A sample the smallest unit in a digital audio, equivalent to pixels in images. An audio is composed of a sequence of samples.

Audio information is stored as an array of bytes. To interpret said array of bytes as samples, and thus a valid audio file, the following terms and parameters, normally stored in the audio header, are relevant:

- **Sample rate:** the number of samples in a second, measured in hertz. A higher sample rate means that the played discretized audio more approaches the original, smooth analog audio. CD quality audio has a sample rate of 44100 Hz.
- **Bit depth**: how many bits comprises a sample. The higher the bit depth, the more accurately the amplitude of an audio is represented. CD quality audio has a bit depth of 16 bits (2 bytes), which means that a sample is represented by a two's complement two-byte integer.
- **Channels**: represents sound coming from or going to a single point. A single microphone produces one channel of audio, and a single speaker accepts one channel of audio. Commonly found audio channels are mono (one channel), stereo (two channels; left and right), and surround sound (six channels).
- **Interleaved**: how samples in different channels are organized. An interleaved audio stores samples belonging to different channels after one another; that is, the first samples of each channel are stored sequentially before the second and the rest. A non-interleaved audio

stores each channel separately; they may be stored in one file and each channel is concatenated after the other, or they may be stored in different files.

Fig. 5. Non-interleaved and interleaved audio with two channels Source: The Lab Book Pages

- **Endianness**: the byte ordering used in storing data types larger than a byte. This is relevant is a sample is composed of more than a byte (8 bits).

F. Lossless Audio Formats

Lossless audio formats encode and store audio data such that the audio is the same as the original source. This contrasts with lossy audio formats such as AAC, MP3, and WMA, which compress audio using algorithms that modifies the original data.

Because lossy formats modify the original audio data, spatial domain manipulation (bits and bytes) of lossy audio data is infeasible; any manipulation must be done in the temporal domain (signals). Lossless audio data preserves the original audio, which also preserves any spatial domain manipulation done to the raw audio data.

III. Solution Analysis and Design

This paper proposes a watermarking scheme combining the EdDSA digital signature algorithm and LSB substitution watermarking. EdDSA is used to generate a digital signature based on a private key and the audio data and used to verify whether the audio data has been tampered at all using a valid public key. LSB substitution watermarking is used to embed the generated signature to the audio data.

LSB watermarking is chosen because of its simplicity. The fragility of LSB watermarking is allowed because if the audio has been tampered, the embedded data will be lost and verification will fail, which is suitable for the purposes of this paper.

Embedding the digital signature to one file eliminates the need of a separate file for authentication. The generated signature is embedded in the audio body, not the header. The watermarking scheme processes audio as raw audio and can be applied to audio stored in a lossless audio format (FLAC, WAV, etc.).

The overall watermarking scheme design (signing and verification scheme) is illustrated in the following diagrams.

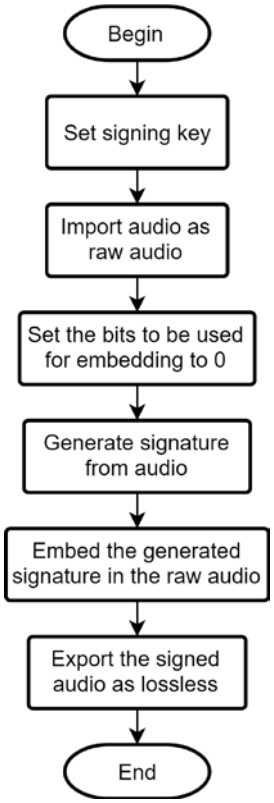

Fig. 6. Block diagram illustration for audio signing

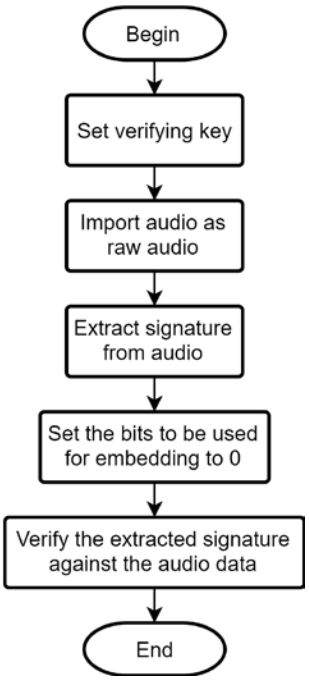

Fig. 7. Block diagram illustration for audio signature verification

Design rationales for the watermarking scheme will be elaborated in the following sections.

A. EdDSA Curve Parameters

As explained in Section II.A, EdDSA requires common curve parameters to be agreed first. This paper uses the Ed25519 signature scheme, which uses the SHA-512 algorithm and the

Curve25519 curve. Ed25519 is represented by the twisted Edwards curve

$$-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2 \quad (8)$$

Ed25519 has the following parameters:

- $p = 2^{255} - 19$ (prime)
- $b = 256$
- Encoding of $GF(p)$: 255-bit little-endian encoding of $\{0, 1, \dots, p-1\}$
- $H(x) =$ SHA-512
- $c = 3$
- $n = 254$
- $d = -\frac{121665}{121666}$
  equivalent to `37095705934669439343138083508754565189542190163887855330859402835551138798432` in $GF(p)$
- $B =$ `(15112221349535400772501151409588531511454012693041857206046113283949847762202, 46316835694926478169428394003475163141307993866256225615783033603165251855960)`
- $\ell = 2^{252} +$ `27742317777372353535851937790883648493`
- $PH(x) = x$

The signature produced is 64 bytes long and stored as the encoded form of (R, S).

B. Modifications to LSB Substitution for Raw Audio

Because a sample in raw audio is usually encoded as a multiple-byte integer, simply replacing the least significant bits of each byte will produce noticeable differences when the audio is played back. This is because each byte in a sample is played as a group and editing the least significant bit in a higher-order byte can produce more noticeable differences than editing the most significant bit in a lower-order byte.

To accommodate, the data can simply be embedded in the least significant bytes of each sample; the first for little-endian systems, the last for big-endian systems. This practically cuts down the embedding capacity by the number of bytes in a sample, but the decrease in capacity does not present a significant impact due to the small size of the embedded data.

The number of samples needed to store the 64-byte signature is 512, which represents about two hundredth of a second in a 44100 Hz 16-bit mono audio.

C. Modifying the Data for Digital Signing

Another problem exists when the digital signature of the audio is going to be embedded: the data after embedding will be different from the data used to make the signature. In which case, the digital signature from the watermarked data will differ from the digital signature embedded.

To overcome this problem, simply set all bits which will be substituted to zero before generating and embedding the digital signature. On verification, set all LSB to zero after extracting the embedded signature, then verify the extracted signature against the raw audio data.

In the watermarking scheme, to simplify, the generated digital signature is embedded in the first 512 samples of the raw audio data, regardless of channels.

IV. Implementation

The proposed watermarking scheme is implemented for proof-of-concept as a Python script. The following external libraries are used to simplify the implementation process.

- `ffmpeg + pydub`: for audio import to raw audio and export to an interchange lossless format
- `gmpy2`: as a multi-precision big integer library
- `hashlib`: provides multiple hash functions

Because data is imported from an interchange format, parameters such as sample rate or bit depth can be read from the header.

The script is divided to three functions: key generation, signing, and verification as detailed below.

- Key Generation
  Usage:
  ```
  eddsa-signer.py keys [-h]
                       [-sk SIGNING_KEY_PATH]
                       [-vk VERIFYING_KEY_PATH]
  ```
- Signing
  Usage:
  ```
  eddsa-signer.py sign [-h]
                       [-i INPUT_PATH]
                       [-o OUTPUT_PATH]
                       [-sk SIGNING_KEY_PATH]
  ```
- Verification
  Usage:
  ```
  eddsa-signer.py verify [-h]
                         [-i INPUT_PATH]
                         [-vk VERIFYING_KEY_PATH]
  ```
  Output: `[True/False]`

The keys are stored as 64-byte binary files. All audio formats supported by `ffmpeg` can be signed, but the resulting file must be saved as an uncompressed or lossless audio format (WAV, FLAC, ALAC, etc.).

V. Testing

There are three security concepts a digital signature is used for: authentication, integrity, and non-repudiation. To that end, the following testing scenarios will be executed:

1. Signing and verifying audio with a correct keypair (control scenario).
2. Changing the public key used for verification.
3. Modifying the watermarked audio data.
4. Modifying the embedded data signature.

There are two points of data noted from testing: execution time and signature verification result. For each file, the signing/verification process is repeated five times to obtain the

average time needed for execution. The signature verification result is considered valid if and only if, in each repetition, the signature is valid.

The following keys are used for testing.

TABLE I. KEYS FOR TESTING

Key	Value		
	base64	***integer***	
Signing key	Hf9MSj7uaU152zKzYKSLJnmcpkNXovh7gPxx7mk8iMo	91607903575316094880636108887987357102399196836832966867528314880766572822301	
Verifying key	OdkAiNDScYOgTgy06OBSxtF9MTAvS54lcA2FeQHrTbI	**x**	-103985244613492019262252526739124125696
		y	2275331160330021937426352718434857940491677958425878746873385658081928969247 3

The following files are used for testing. Only WAV files are used for testing because WAV files are raw audio with a RIFF header, so the time measured is outside the conversion from an interchange format (FLAC, lossy audio) to and from raw audio.

TABLE II. FILES FOR TESTING

Duration (ms)	Size (KB)
100	19
1000	188
60000	11251
271728 (full)	50950

Testing is done on a machine with the following specification.

- Operating System: Windows 10 Home 64-bit (10.0, Build 16299)
- Processor: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz (4 CPUs), ~2.9GHz
- Memory: 8192MB RAM
- Hard Drive: 1TB NTFS
- Python Interpreter: CPython 3.6.3

A. *Control Scenario*

The control scenario describes a valid signing/verifying flow; that is, a matching pair of signing and verifying key is used on a valid audio data unchanged between the signing and verification process.

TABLE III. TEST RESULT FOR CONTROL FLOW

Duration	Execution time (ms)		Verification result	
	Signing	***Verifying***	***Expected***	***Actual***
100	15.625	15.625	**Pass**	**Pass**
1000	15.625	15.625	**Pass**	**Pass**
60000	162.5	96.875	**Pass**	**Pass**
271728	643.75	418.75	**Pass**	**Pass**

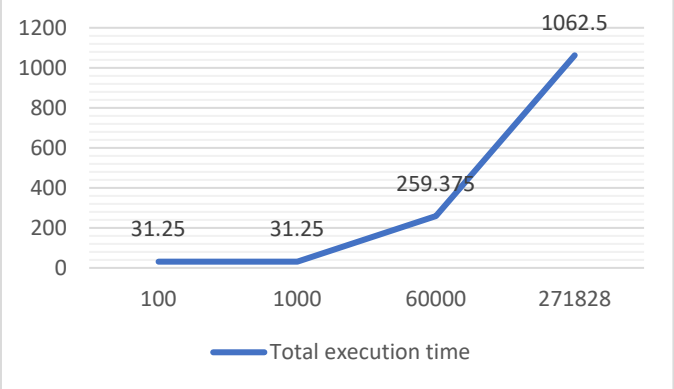

Fig. 8. Total executon time of signing/verifying process in the control scenario

B. *Keypair Mismatch*

In this scenario, the verifying key in Table I is swapped with the verifying key below. This scenario tests the watermarking scheme authentication; that is, a mismatched keypair should not return a positive (valid) result.

TABLE IV. KEYS FOR TESTING KEYPAIR MISMATCH

Key	Value		
	base64	***integer***	
Verifying key	rX2aRBRHQSBpsVPKbRB/3Xkvf5FL7YGuRO3kmqoynmA	**x**	20661558547513833432605821887815929036 8
		y	43701544987049098903874852211172298850732823836676555728552433950999698898349

TABLE V. TEST RESULT FOR KEYPAIR MISMATCH

Duration	Execution time (ms)		Verification result	
	Signing	***Verifying***	***Expected***	***Actual***
100	15.625	15.625	**Fail**	**Fail**
1000	15.625	15.625	**Fail**	**Fail**
60000	140.625	81.25	**Fail**	**Fail**
271728	646.875	375.0	**Fail**	**Fail**

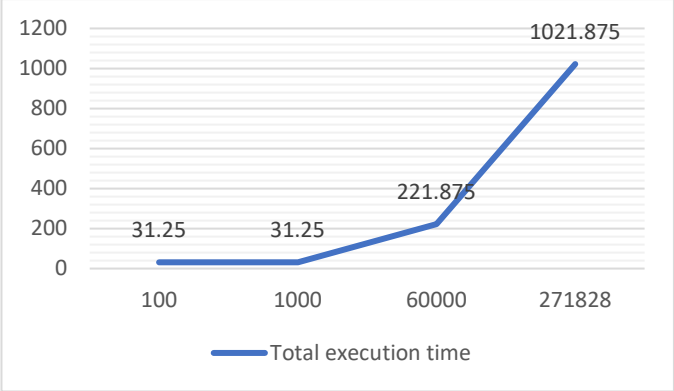

Fig. 9. Total executon time of signing/verifying process in the keypair-mismatch scenario

C. Modified Data

In this scenario, the audio data is modified after the audio has been watermarked. This is done by changing the bits of the first byte of the raw audio data to zero, except the LSB (to preserve the signature). This scenario tests for integrity; that is, a corrupt file should not return a positive (valid).

TABLE VI. TEST RESULT FOR MODIFIED DATA

Duration	Execution time (ms)		Verification result	
	Signing	*Verifying*	*Expected*	*Actual*
100	15.625	15.625	**Fail**	**Fail**
1000	15.625	15.625	**Fail**	**Fail**
60000	168.75	90.625	**Fail**	**Fail**
271728	656.25	381.25	**Fail**	**Fail**

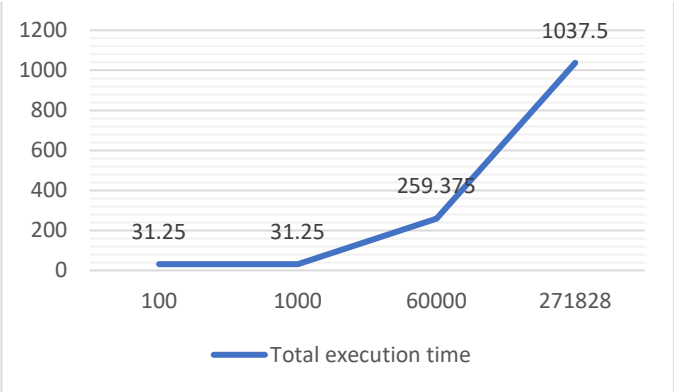

Fig. 10. Total executon time of signing/verifying process in the modified-data scenario

D. Modified Signature

In this scenario, the embedded signature itself is modified. This is done by flipping the LSB where the first bit of the signature is embedded. This scenario tests for integrity; that is, an invalid signature should not return a positive (valid).

TABLE VII. TEST RESULT FOR MODIFIED SIGNATURE

Duration	Execution time (ms)		Verification result	
	Signing	*Verifying*	*Expected*	*Actual*
100	15.625	15.625	**Fail**	**Fail**
1000	15.625	15.625	**Fail**	**Fail**
60000	150.0	103.125	**Fail**	**Fail**
271728	659.375	425.0	**Fail**	**Fail**

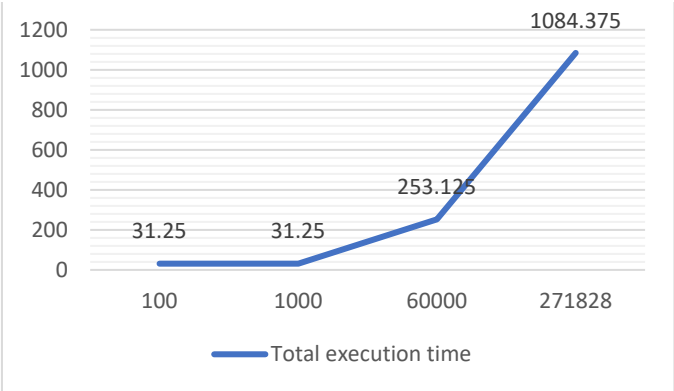

Fig. 11. Total executon time of signing/verifying process in the modified-signature scenario

VI. ANALYSIS

A. Performance

Testing results show that the signing and verifying process execution time has a lower bound of 15.625 milliseconds. This is caused by the overhead in importing and exporting audio, as well as loading the internal processes involved, such as loading the libraries involved.

For larger files, the execution time increases linearly with the audio file size. In general, the signing process takes about 1.5 times longer than the verification process. This may be caused by the need to import, hash, and export the watermarked audio file.

B. Correctness

Judging by the verification result, it can be concluded that using EdDSA for signing lossless audio is feasible. The signature signing and verification method correctly identified tampered audio files and signature, thus fulfilling the integrity aspect of security, and verifies the signer of the audio file, fulfilling the authentication and nonrepudiation aspect of security.

C. Robustness

Due to inherent limitations on operating on the spatial domain, the watermarking scheme is very much fragile. Any modification to the original data or the embedded signature itself will invalidate the embedded signature. This renders signing more common lossy audio formats (MP3, AAC, OGG, etc.) impossible.

A more robust signature embedding would operate in the temporal domain and manipulate audio as signals, instead of bits and bytes. Embedding in the temporal domain would make signing lossy formats possible, but is significantly more difficult to do than simply manipulating bits and bytes.

VII. Conclusion

Three points of conclusion can be inferred for this paper:

1. An EdDSA digital signature can be used to protect an audio file in three security concepts: authentication, integrity, and non-repudiation.
2. The proposed watermarking scheme is simple to use and can be used to generate, embed, and verify digital signature for all lossless audio formats without any further need for additional files or tags and without significantly changing the watermarked audio.
3. This scheme can further be improved by changing the watermarking method used; of note is temporal (signal) manipulation.

References

[1] H. Hisil, K. K.-H. Wong, G. Carter and E. Dawson, "Twisted Edwards Curves Revisited," *Advances in Cryptology - ASIACRYPT 2008 Lecture Notes in Computer Science*, pp. 326-343, 2008.

[2] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe and B.-Y. Yang, "High-speed high-security signatures," *Cryptographic Hardware and Embedded Systems – CHES 2011 Lecture Notes in Computer Science*, pp. 124-142, 2011.

[3] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," January 2017. [Online]. Available: https://tools.ietf.org/html/rfc8032. [Accessed 15 April 2018].

[4] Y. Romalier and S. Pelissier, "Practical fault attack against the Ed25519 and EdDSA signature schemes," *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2017.

[5] T. Lange, "Extended coordinates with a=-1 for twisted Edwards curves," hyperelliptic, [Online]. Available: https://hyperelliptic.org/EFD/g1p/auto-twisted-extended-1.html. [Accessed 18 May 2018].

[6] D. J. Bernstein, P. Birkner, T. Lange and C. Peters, "ECM using Edwards curves," *Mathematics of Computation*, vol. 82, no. 282, pp. 1139-1179, 2012.

[7] B. Warner, "How do Ed5519 keys work?," Mozilla Blog, 29 November 2011. [Online]. Available: https://blog.mozilla.org/warner/2011/11/29/ed25519-keys/. [Accessed 17 May 2018].

[8] D. Connor, "Sample Rate and Bitrate: The Guts of Digital Audio," The Stereo Bus Blog, [Online]. Available: https://thestereobus.com/2008/01/12/sample-rate-and-bitrate-the-guts-of-digital-audio/. [Accessed 17 May 2018].

[9] E. L. Blake, "Raw Audio File Formats Information," Fmtz, [Online]. Available: https://web.archive.org/web/20160525083851/http://www.fmtz.com/misc/raw-audio-file-formats. [Accessed 17 May 2018].

[10] J. Robert, *Pydub*, GitHub, 2018.

[11] R. Munir, Diktat Kuliah IF5054 Kriptografi, Bandung: Departemen Teknik Informatika Institut Teknologi Bandung, 2005.

Declaration

I hereby declare that this paper is my own writing, not an adaptation, translation, or plagiarized from others' paper.

Bandung, May 17, 2018

Felix Limanta 13515065