**e\_sign tool User Guide**

1. **OVERVIEW:**

This document provides necessary instruction and steps to generate keypair and signature and eventually append the electronic signature (e-sign) to an input binary image.

1. **ALGORITHM:**

* **Cryptography**: ECDSA-NIST256p (a.k.a. ECDSA-SECP256r1)
* **Hashing**: SHA-256

1. **PYTHON CRYPTOGRAPHIC LIBRARY USED:**
   * **ecdsa**: This is an easy-to-use implementation of ECDSA cryptography (Elliptic Curve Digital Signature Algorithm), implemented purely in Python, released under the MIT license.
2. **Command description*:***

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Value to pass** | **Usage** | **Description** |
| -h/ --help |  | -h / --help | Displays help for this python script and exits |
| -genkey /  --gen\_key | keypair | -genkey keypair/  --gen\_key keypair | Generation of key-pair files  (Private.bin and Public.bin) |
| -dir | <output-directory path> | -dir <output-directory path> | Generates file(s) in this directory. |
| -hash\_gen | hash | -hash\_gen hash | Generates hash of the binary-image and displays the hash on the console window. |
| -i / --input | <input binary-image> | -i <input\_binary-image> /  --input <input binary-image> | Input binary-image file. |
| -gen\_sign | sign | -gen\_sign sign | Generates signature file  (signature.bin). |
| -private\_key | <input private-key file> | -private\_key <input private-key file> | Input private-key file.  (Private.bin) |
| -public\_key | <input public-key file> | -public\_key <input public-key file> | Input public-key file.  (Public.bin) |
| -verify | verify | -verify verify | Verifies input signature file against the input public-key file and input binary-image. |
| -append | append | -append append | Overwrites the input binary image and appends the signature to it. |

1. **Command Usage:**
2. Generate key-pair (Private and Public Key):

**Output:**

* Files: “Private.bin” and “Public.bin” are generated at <output-directory path>

<output-directory path> -> directory where key-pair is generated.

“Private.bin" -> Private-key file.

“Public.bin" -> Public-key file.

**Command:**

* e\_sign.exe -genkey keypair -dir <output-directory path>

1. Generate Hash: *(optional)*

**Output:**

* Hash (SHA-256) of the <input binary-image> is generated and displayed on the console**.**

<input binary-image> -> input binary-image file

**Command:**

* e\_sign.exe -hash\_gen hash -i <input binary-image>

1. Generate signature:

**Output:**

* Signature file (signature.bin) of <input binary-image> is generated at <output-directory path> using <input private-key>.

<output-directory path> -> directory where signature file is generated.

<input binary-image> -> input binary-image file.

<input private-key> -> input private-key file.

**Command**:

* e\_sign.exe -gen\_sign sign -i <input binary-image> -private\_key <input private-key> -dir <output-directory path>

1. Signature verification: *(optional)*

**Desired Output:**

* "Verified OK!" -> Verification Successful.

<input signature-file> is checked against <input binary-image> and <input public key-file>.

<input binary-image> -> input binary-image file

<input signature-file> -> input signature-file

<input public key-file> -> input public-key file

Command:

* e\_sign.exe -verify verify -i <input binary-image> -signature <input signature-file> -public\_key <input public key-file>

1. Appending signature:

**Output:**

* <input binary-image> is overwritten and <input signature-file> is appended to it.

<input binary-image> -> input binary-image file.

<input signature-file> -> input signature-file.

**Command**:

* e\_sign.exe -append append -i <input binary-image> -signature <input signature-file>