1 Context

TrustID is an IDP, which stands for a (digital) IDentity Provider that can have various uses, such as 2-Factor Authentication (2FA) or digital signatures. As such, TrustID has to offer ways to securely identify, verify, and authenticate users. To achieve this, TrustID has a strong onboarding procedure, either using a video onboarding process or by having users physically go to an onboarding office, as well as providing an authenticator application for the user's mobile phone. Once onboarded, a user that wants to access a service secured by TrustID enter their credentials ([USERNAME] and password) on the TrustID website. The TrustID server then sends a challenge to the authenticator application, which then uses the unlocking mechanism of the phone to authentify the user: we assume only the targeted user can unlock their phone. The widespread use of biometrics as a way of unlocking phones (rather than traditional passwords or PINs) would ideally improve the security of this method drastically. However, unlocking processes typically offer one of the weaker methods as a fallback. Therefore, TrustID relies only on the ownership of the phone for security.

That brings us to our security issue: at time of the internship, When the user validates a [TRANSACTION] on their mobile phone (i.e press the "accept" button of the authenticator app), they have no way of being sure that the prompt on their phone corresponds to the actual [TRANSACTION] they wish to effectuate. This could lead to some vulnerabilities: for example, an impersonation attack is possible, as shown here (we take the example of digital signatures for illustration purposes):

Say an an attacker A wants to sign a document we will call d_A while impersonating user U. U uses a weak password, which means A can log as U on the TrustID website.

- A logs as U on TrustID website.
- A waits for U to ask for a challenge for any document.
- When U asks for a challenge for some document d_U, A asks the server for a challenge for d_A.
- The server will then send two challenges to U's phone, one for d_A and one for d_U.
- Since U is unable to distinguish between the two challenges, they validate whichever challenge comes first.
- If the challenge for d_A came first, then the attacker successfully signed a document while impersonating U.

While not that impactful when used on digital signatures, the same attack could be used for other (much more sensitive) TrustID services, such as 2FA. This project is aimed at solving the security issue presented above. The way we will achieve it is to present the user with a unique, recognizable fingerprint of the document they requested a challenge for. While this could be achieved with traditional hash functions, the fact that human users will have to compare fingerprints calls for a more user-friendly solution. In particular, we want the fingerprints to have the form of pictures, as we hypothesize they are less cumbersome to compare.

Note: in practice, A's and U's requests for challenges are augmented with metadata. Different metadata calls for different capabilities of A: a UserID would not call for anything more than assumed earlier. A timestamp (currently in use at time of writing) requires A to send their request simultaneously with U. A nonce would require A to guess the nonce. [MAYBE :] We will see [later] how the choice of metadata alter the probability of a successful attack.

2 Visual Hashing

2.1 Definitions and notations

2.1.1 Pixels

Because we work on image representation, we will work with pixels. We write $\mathbb{P}$ the set of all pixels. We will often need separate categories of pixels: in particular, we write $\mathbb{P}_2 \subset \mathbb{P}$ the set of pixel that may only take two values (namely "white" and "black"). We write $\mathbb{P}_{\mathsf{RGB}} \subset \mathbb{P}$ the set of pixels that are encoded by three color channels (Red, Green, Blue), each channel having an integer value between 0 and 255.

2.1.2 Visual hash function

We define a visual hash function as a function $\{0,1\}^* \to \mathbb{P}^{m\times n}$ of the form:

$$\mathcal{H}(x;\lambda) = V(H(x;\lambda);\lambda)$$

Where λ is a security parameter, $H : \{0,1\}^* \to \{0,1\}^\ell$ is a standard hash function (such as $\mathsf{SHA-256}$) and $V : \{0,1\}^\ell \to \mathbb{P}^{m\times n}$ is a mapping from bits strings to arrays of pixels. The values of ℓ, m, n depend on the value of λ.

Whenever it is not explicitly needed, we omit the security parameter λ in the notation. We call $h := H(x)$ a *hash*, and $F := V(h)$ a *fingerprint*. We note a visual hash function as $\mathcal{H} = (H, V)$ or when contexts allows it, just $\mathcal{H}$.

2.1.3 Visual indistinguishability

For two fingerprints $F_1, F_2 \in \mathbb{P}^{m\times n}$, we write $F_1 \approx_v F_2$ if F_1 is *visually indistinguishable* from F_2, that is, if a user thinks $F_1 = F_2$ by looking at them. Note that $F_1 = F_2 \implies F_1 \approx_v F_2$. [TO BE CLARIFIED / EXPANDED IN DEDICATED SECTION]

2.1.4 Properties of visual hash functions

We adapt traditional hash functions properties to visual hash functions as follows [NOT SURE IF USEFUL YET]. When we say that an attack is "infeasible", we mean that an efficient [DEFINE EFFICIENT - PROBABLY POLY-TIME] adversary cannot realistically perform the attack because of its complexity. The properties are as follows:

- *First pre-image resistance*: we say a visual hash function $\mathcal{H}$ is first pre-image resistant if given $F \in \mathbb{P}^{m\times n}$, it is infeasible to find $x \in \{0,1\}^*$ such that $\mathcal{H}(x) \approx_v F$.
- *Second pre-image resistance*: we say a visual hash function $\mathcal{H}$ is second pre-image resistant if given $y \in \{0,1\}^*$, it is infeasible to find $x \neq y \in \{0,1\}^*$ such that $\mathcal{H}(x) \approx_v \mathcal{H}(y)$.
- *Collision resistance*: we say a visual hash function $\mathcal{H}$ is collision-resistant if it is infeasible to find $x, y \in \{0,1\}^*$ such that $x \neq y$ and $\mathcal{H}(x) \approx_v \mathcal{H}(y)$.

In order to better categorize and analyze visual hash function, we define two additional properties:

- *Intermediate pre-image resistance*: we say a visual hash function $\mathcal{H} = (H, V)$ is intermediate pre-image resistant if given $F \in \mathbb{P}^{m\times n}$, it is infeasible to find $h \in \{0,1\}^\ell$ such that $V(h) \approx_v F$.
- *Visual injectiveness*: we say a visual hash function $\mathcal{H} = (H, V)$ is visually injective if for any $h_1, h_2 \in \{0,1\}^\ell$, $h_1 \neq h_2 \implies V(h_1) \not\approx_v V(h_2)$. Note that visual injectiveness implies that V is injective, because $V(h_1) \not\approx_v V(h_2) \implies V(h_1) \neq V(h_2)$

3 Context of application

3.1 Protocol

We consider the protocol shown in fig. 1, in which a visual hash function $\mathcal{H} = (H, V)$ is used. On the server side, md denotes some metadata such as a sessionID, timestamp, nonce, etc. [TO BE EXPANDED IN IMPLEMENTATION SECTION].

3.2 Threat models

3.2.1 A realistic TrustID threat

[GIVE IDEA OF TRANSACTION VALIDATION PROTOCOL (perhaps simplified version of Antoine's UML sequence diagram)]. Following [that], we assume an adversary cannot impersonate the TrustID server. However, an attack by an external adversary is still possible. We define, for an adversary $\mathcal{A}$, the following game:

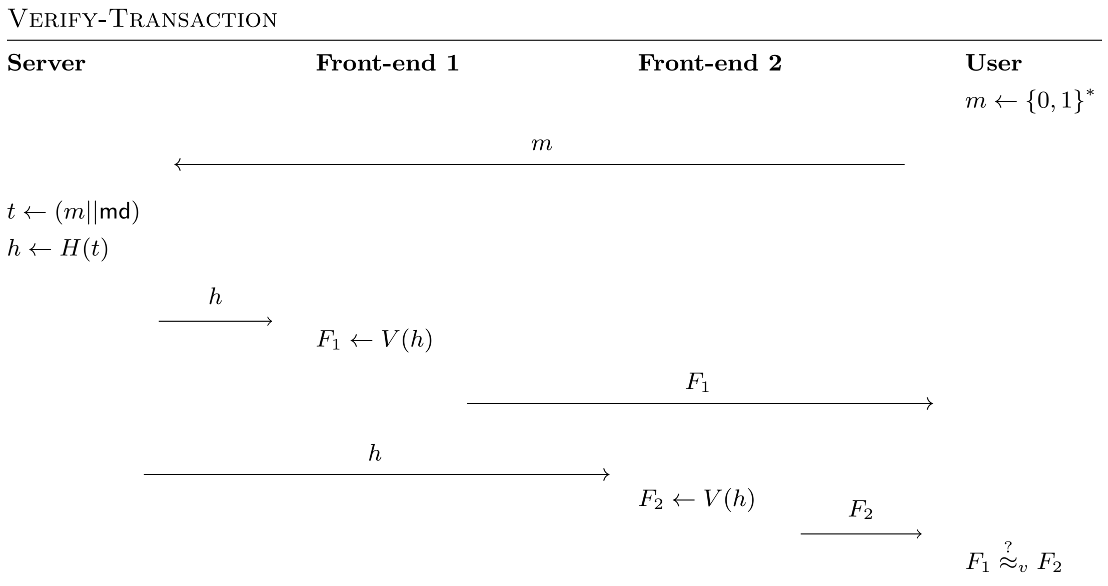

Figure 1: Protocol without adversary.

```
Game $\Gamma_{\mathsf{TrustID}}$
$m \leftarrow \{0,1\}^*$
$t \leftarrow (m||\mathsf{md})$
$h \leftarrow H(t)$
$F \leftarrow V(h)$
$\mathcal{A}(t^*) \rightarrow m'$
$t' \leftarrow (m'||\mathsf{md}')$
$h' \leftarrow H(t')$
$F' \leftarrow V(h')$
**return** $1_{F \approx_v F' \ \wedge \ m' \neq m}$
```

Where t^* denotes some part of t: depending on the metadata we choose, $\mathcal{A}$ may know some bits of t that are public (e.g, a timestamp or the user's ID). We consider our scheme achieves the minimal required security if for any efficient adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\Gamma_{\mathsf{trustid}}}(\lambda)$ is negligible in λ, where $\mathsf{Adv}_{\mathcal{A}}^{\Gamma_{\mathsf{trustid}}}(\lambda) = \Pr[\Gamma_{\mathsf{TrustID}}$ returns 1]. Figure 2 illustrates how the game translates to a concrete implementation.

While we could try and design some visual hash function that achieves this minimal security, we choose to go further and define a stronger threat model: this could alleviate our assumptions on the security of the rest of the TrustID architecture, as well as provide further insight.

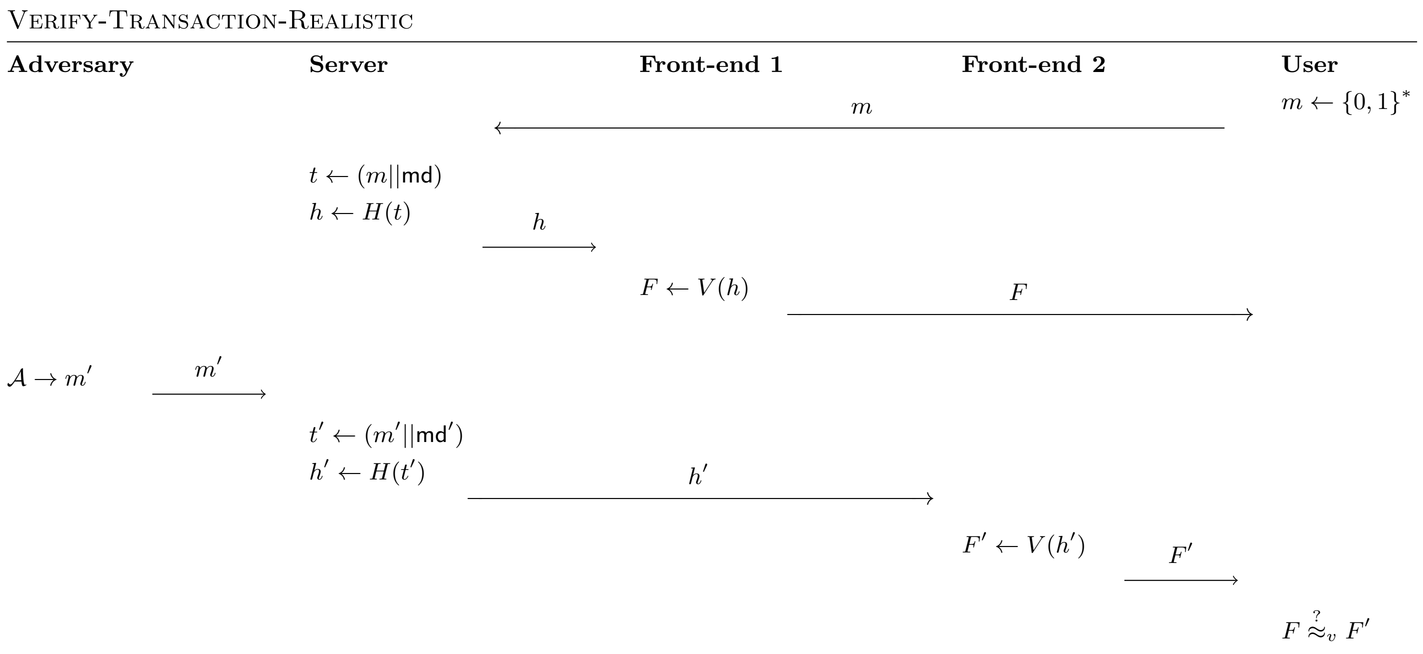

Figure 2: Protocol with a realistic TrustID adversary.

3.2.2 A stronger threat

We now define another game, with an adversary $\mathcal{A}$:

```
Game Γ
$m \leftarrow \{0,1\}^*$
$t \leftarrow (m||\mathsf{md})$
$h \leftarrow H(t)$
$F \leftarrow V(h)$
$\mathcal{A}(t, h, F) \rightarrow h'$
$F' \leftarrow V(h')$
return $1_{F \approx_v F' \ \wedge \ h' \neq h}$
```

In words, $\mathcal{A}$ wins if they can produce a hash h', different from the true hash h, that yields a fingerprint $V(h')$ that is visually indistinguishable from the true fingerprint F. We will consider our scheme is secure if for any efficient adversary $\mathcal{A}$, $\mathsf{Adv}^{\Gamma}_{\mathcal{A}}(\lambda)$ is negligible in λ, where $\mathsf{Adv}^{\Gamma}_{\mathcal{A}}(\lambda) = \Pr[\Gamma \text{ returns } 1]$

We show that this security is indeed stronger that the minimal security we defined in the previous section. Concretely, we prove that, $\mathsf{Adv}^{\Gamma}_{\mathcal{A}}(\lambda)$ is negligible for all efficient adversaries $\implies$ $\mathsf{Adv}^{\Gamma_{\mathsf{trustid}}}_{\mathcal{A}}(\lambda)$ is negligible for all efficient adversaries. We proceed by reduction: suppose $\mathcal{A}_{\mathsf{TrustID}}$ is an efficient adversary playing $\Gamma_{\mathsf{TrustID}}$ that wins with non-negligible probability. We build an adversary $\mathcal{A}$ playing Γ that uses $\mathcal{A}_{\mathsf{TrustID}}$ to win as follows:

```
$\mathcal{A}(t, h, F)$
Extract $t^*$ from $t$ depending on what $\mathcal{A}_{\mathsf{TrustID}}$ needs
$\mathcal{A}_{\mathsf{TrustID}}(t^*) \rightarrow m'$
$t' \leftarrow (m'||\mathsf{md}')$
$h' \leftarrow H(t')$
return $h'$
```

Note that we suppose that $\mathcal{A}$ has a way of efficiently knowing md'. Now, if $\mathcal{A}_{\mathsf{TrustID}}$ wins at $\Gamma_{\mathsf{TrustID}}$, then $\mathcal{A}$ wins at Γ. Therefore, $\mathsf{Adv}^{\Gamma}_{\mathcal{A}}(\lambda) \geq \mathsf{Adv}^{\Gamma_{\mathsf{trustid}}}_{\mathcal{A}_{\mathsf{TrustID}}}(\lambda)$, which was assumed to be non-negligible, which means $\mathsf{Adv}^{\Gamma}_{\mathcal{A}}(\lambda)$ is non-negligible. That concludes the proof: if there exists an efficient adversary that breaks $\Gamma_{\mathsf{TrustID}}$, then there exists an efficient adversary that breaks Γ. This is equivalent to the statement we wanted to prove.

Figure 3 illustrate how the threat is stronger: the adversary can bypass the server to submit a hash directly to the frontend.

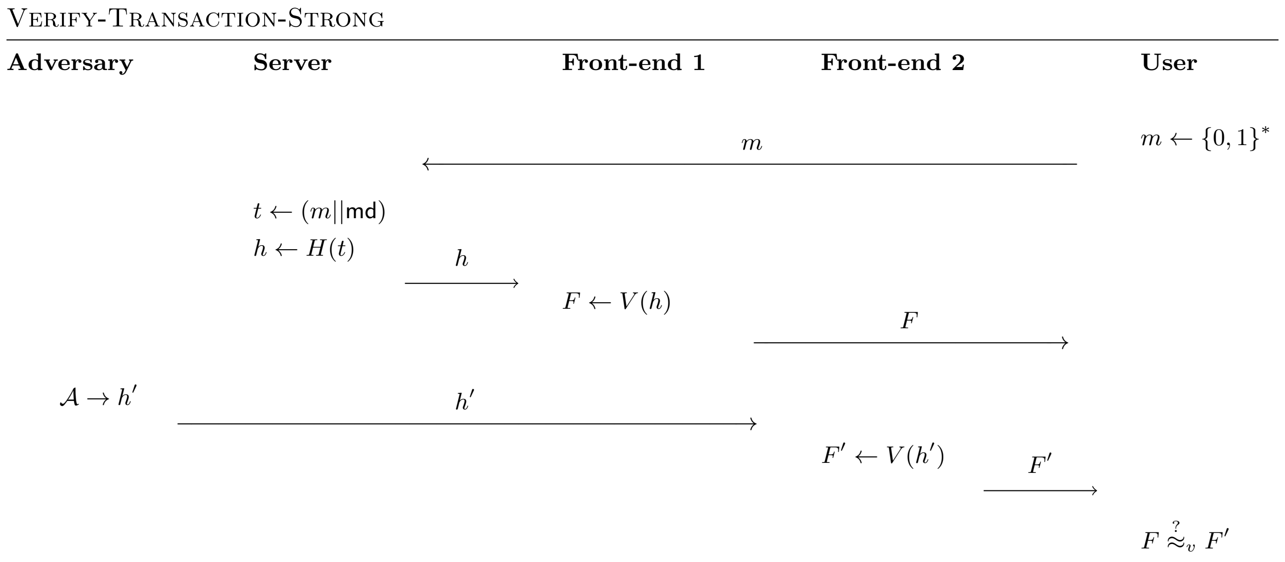

Figure 3: Protocol with a stronger adversary.

4 Security implications of visual hash function properties

We separate the different combinations of properties a visual hash function may have, to analyse their strengths and weaknesses in terms of security.

4.1 No visual injectiveness, no intermediate pre-image resistance

If a visual hash function $\mathcal{H}$ is not intermediate pre-image resistant and isn't visually injective, the adversary can create a forgery:

1. From F, find $F' \neq F \in \mathrm{Im}(V)$ such that $F \approx_v F'$.
2. From F', get h' such that $V(h') = F'$.
3. Output h'.

This attack is efficient by assumption as long as the adversary can find F' efficiently. Therefore we need at least one of the two proprieties to keep the adversary from making forgeries, or somehow make it infeasible to find F' from F.

4.2 Visual injectiveness

If $\mathcal{H} = (H, V)$ is visually injective, then the adversary never wins: indeed, by definition of visual injectiveness, $h \neq h' \implies V(h) \not\approx_v V(h')$.

Moreover, visual injectiveness implies that the only case where $V(x_1) \approx_v V(x_2)$ with $x_1 \neq x_2$ is when $H(x_1) = H(x_2)$. That is, in order to find a collision on $\mathcal{H}$, the adversary must find a collision on H. Theferore the following implication holds:

$$\mathcal{H} \text{ is visually injective} \wedge H \text{ is collision-resistant} \implies \mathcal{H} \text{ is collision-resistant} \tag{1}$$

Since there are well-documented hash functions that are considered collision-resistant, visual injectiveness would be the ultimate goal for our hash function. However we will (probably) see that visual injectiveness is hard to achieve if the input space of V is large (i.e, ℓ is large)

4.3 Intermediate pre-image resistance

If $\mathcal{H} = (H, V)$ is intermediate pre-image resistant but isn't visually injective, the attack described in section 4.1 is not feasible, because step 2 is infeasible by intermediate pre-image resistance. [BUT IS IT ENOUGH FOR SECURITY ?]

5 On visual indistinguishability

5.1 Evaluating the difference between fingerprints

We would like a metric to measure how well our scheme performs, so that we can analyse it formally. In particular, we need to define a distance between d between two fingerprints $F_1, F_2 \in \mathbb{P}$ such that for some $\delta > 0$ (ideally as small as possible)

$$d(F_1, F_2) \leq \delta \iff F_1 \approx_v F_2$$

5.1.1 normalised Hamming distance

A natural choice for evaluating the differences between binary images (i.e, that are in $\mathbb{P}_2^{m\times n}$) would be to compute the normalised Hamming distance of the two fingerprints

$$d_H(F_1, F_2) = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \mathbb{I}(F_1[i][j] \neq F_2[i][j])$$

where $F_1, F_2 \in \mathbb{P}^{m\times n}$, $(\cdot)[i][j]$ denotes the pixel at position (i, j), and $\mathbb{I}$ is an indicator function. d_H ranges between 0 and 1 included, the extrema being obtained by having two pictures that differ in zero or all of their pixels, respectively. This metric would be useful because it relates directly to the Hamming distance between of the hashes functions.

Unfortunately, it seems this distance does not achieve the properties we stated above. We can refute the right direction ($\implies$) very easily. Indeed, fig. 4 shows two fingerprint with the smallest non-zero Hamming distance ($d_H = \frac{1}{mn}$, $mn = 256$) that are clearly visually distinguishible by humans. Concerning the other direction ($\impliedby$), we should proceed with caution: shifting bits can produce images that look similar despite having a large Hamming distance. Figure 5 shows fingerprints having a normalised Hamming distance of $\frac{71}{256} = 27.73\%$ yet the results are not easy to distinguish.

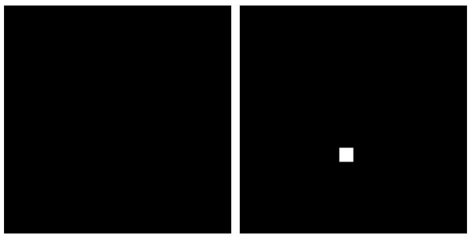
Figure 4: Two visually distinguishable fingerprints with low Hamming distance.

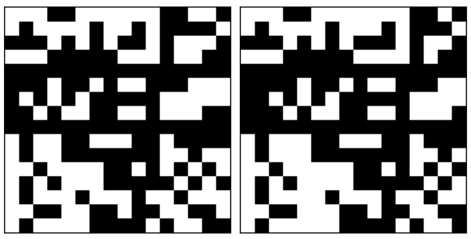
Figure 5: Two fingerprints that have a large Hamming distance, but are not trivial to distinguish.

5.1.2 HaarPSI distance

The Haar perceptual similarity index (HaarPSI) [ref] seems to be a relatively good measure to test whether a pair of fingerprints is distinguishable. The advantages it offers are a relatively low computation time compared to other PSI, which allows for automated parameter search, as well as a good correlation with human opinion scores [ref]. In contrast, the main downside is that although the results are re-linearized, the final output (a number between 0 and 1) is somewhat tricky to interpret: two clearly distinguishable images will have a HaarPSI around 0.1–0.2, and a HaarPSI over 0.6 yields a visual difference that is so small that we consider it imperceptible.

For our future scheme, we would of course like to minimize the HaarPSI distance between any 2 fingerprints. However, given the complexity of the HaarPSI formula, we cannot work our way from that to create a scheme. Instead, we pick a threshold value $T_{Haar} = 0.5$ and we consider that

$$\mathsf{HaarPSI}(F_1, F_2) \geq T_{Haar} \implies F_1 \approx_v F_2$$

The value of T_{Haar} was picked arbitrarily: indeed, determining the value experimentally would require user testing on many subjects, a process for which this project has neither the ambition nor the resources. Note that the condition is necessary but not sufficient: indeed, we empirically found some pairs of fingerprints with a low PSI distance that are not easy to distinguish.

6 Existing schemes

6.1 Antoine's

Antoine's visual hash is a very simple scheme: H is the standard SHA-256 function, and V is a trivial bit-to-binary pixel mapping (a bit of value 1 gets mapped to a pixel of value 1 (white), a bit of value 0 gets mapped to a pixel of value 0 (black)). It is clear that this scheme is not intermediate pre-image resistant, as the trivial mapping can easily be reversed. Moreover, it is not sparse, as any combination of 256 binary pixels is a valid output: from a fingerprint F one can produce a fingerprint F' that differs from F in only one pixel, which means $F \approx_v F'$ in general. Therefore, we can conduct the attack described in section 4.1 to create a forgery.

6.2 LifeHash

The LifeHash library [ref] employs the standard SHA-256 hash function, but it introduces some complexity in the definition of b using John Conway's Game of Life [ref]. Let $\mathcal{G} : \mathbb{P}^{m \times n} \to \mathbb{P}^{m \times n}$ be the result of a single generation of the game of life rules on the input. We suppose fingerprints support element-wise addition, as well as multiplication by a constant.

Algorithm 1 describes the working of the LifeHash visual hashing function.

```
**Data:** $x \in \{0,1\}^*$
**Result:** $F \in \mathbb{P}^{m_n}_{\mathsf{RGB}}$
**begin**
    $B \longleftarrow \mathsf{Antoine}(x)$
    Initialize $G$ as the empty array
    **while** $|G| < 150$ ***and*** *Game of Life has not converged* **do**
        Append $B$ to $G$
        $B \leftarrow \mathcal{G}(B)$        ▷ One iteration of Game of Life rules, 1 is alive, 0 is dead
    Initialize $F$ as an all-white (255,255,255) fingerprint the same size as $B$
    **for** $i = 0, \dots, |G| - 1$ **do**
        $F = F - G[i] * \frac{255i}{|G|}$        ▷ Aggregate all generations
    Apply RGB mapping to $F$, using some of the first bits to pick a color scheme.
    Apply symmetry to $F$, using some of the first bits to pick a transformation scheme.
    **return** $F$
```

Algorithm 1: LifeHash

Due to the unpredictable nature of the Game of Life, one would be tempted to think that this scheme is intermediate pre-image resistant: since one can construct a universal Turing machine in the Game of Life [ref], being able to predict for an arbitrary h the final state of the Game of Life (or even whether the game of life converges in 150 generations)

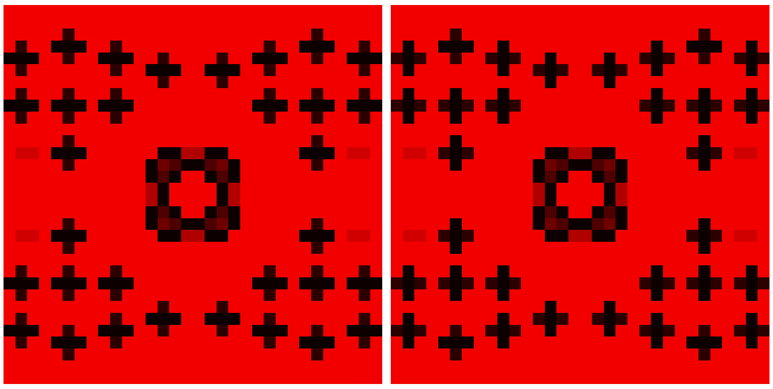
Figure 6: LifeHash outputs when $H = \hat{h}_1$ (left) and $H = \hat{h}_2$ (right)

can be reduced to the Halting problem [ref]. But while it may be intermediate resistant on complex inputs, the Game of Life has known, simple patterns that oscillate, which might cause problems if the input hash is simple (i.e. has a low Hamming weight). We describe an example of a forgery in order to illustrate that this scheme is neither intermediate pre-image resistant nor visually injective:

Let the initial state of the Game of Life B_1 contain a known oscillator. Let the following state sequence when applying the Game of Life rules successively:

$$B_1 \xrightarrow{\mathcal{G}} B_2 \xrightarrow{\mathcal{G}} B_3 \xrightarrow{\mathcal{G}} \ldots \xrightarrow{\mathcal{G}} B_1$$

Now, say we have two different input x_1 and x_2 such that $\mathsf{Antoine}(H(x_1)) = B_1$ and $\mathsf{Antoine}(H(x_2)) = B_2$. Then the execution of LifeHash will go through the same sequence of Game of Life (although rotated one state left for the second), yielding visually similar results. Figure 6 shows the output picture on input x_1 and x_2 when

$$h(x_1) := \hat{h}_1 = \texttt{0000000000000e00e0e0000e00000000eee00000000000006e030002000e0008}$$
$$h(x_2) := \hat{h}_2 = \texttt{0000000000004e404044404400044440444044400000004006403040200 0e0008}$$

Both hashes contain the same period-3 oscillator (with the same initial state) in the center, and a collection of period-2 oscillators around it: 3 horizontally aligned pixels will flip to 3 vertically aligned pixels back and forth, thus giving a cross pattern on the fingerprint. Both need $\text{lcm}(3, 2) = 6$ generations of the Game of Life to converge. The fingerprints differ only in the initial direction of the period-2 oscillators: on the left they start horizontally, whereas on the right they start vertically. Because each period-2 oscillators go through 3 entire cycles, the difference between the two setups becomes unnoticeable.

This pattern is really simple, but it is hard to determine whether other more complicated examples of oscillators combinations yield similar results.

6.3 PRG-based libraries

Some visual hash function use Pseudo-Random Generators (PRG) to create fingerprints. The idea is to seed a PRG with the input value, and let the PRG's randomness act to create fingerprints. Most implementations of such schemes use them directly on the messages (as opposed to hashes), which is not achievable in our case. However, they may have some desirable properties that we analyze in the following sections.

6.3.1 Blockies

Blockies [ref] is a light-weight javascript library used to create icons. It is at the time of writing quite popular (more than 1500 weekly downloads according to the official website), following the gain in popularity of cryptocurrency exchange and the need to identify 42-hexadecimal-characters Ethereum addresses.

Blockies is based on a custom-made PRG (!) based on the method used by Java to calculate hashCodes

```
Data: $x \in \{0,1\}^*$
Result: $F \in \mathbb{P}_{\mathsf{RGB}}^{m \times n}$
begin
    $\mathsf{PRG}_{blockies}.\mathsf{Seed}(x)$    // $\mathsf{PRG}_{blockies}$ returns values between 0 and 1

    $colorBack \longleftarrow (\mathsf{PRG}_{blockies}.\mathsf{next}() * 255, \mathsf{PRG}_{blockies}.\mathsf{next}() * 255, \mathsf{PRG}_{blockies}.\mathsf{next}() * 255)$
    $colorFront \longleftarrow (\mathsf{PRG}_{blockies}.\mathsf{next}() * 255, \mathsf{PRG}_{blockies}.\mathsf{next}() * 255, \mathsf{PRG}_{blockies}.\mathsf{next}() * 255)$
    $colorSpots \longleftarrow (\mathsf{PRG}_{blockies}.\mathsf{next}() * 255, \mathsf{PRG}_{blockies}.\mathsf{next}() * 255, \mathsf{PRG}_{blockies}.\mathsf{next}() * 255)$

    Initialize $F$ as an all-zero fingerprint of size $m \times n$
    foreach pixel $p$ in $F$ such that $p$ is in the left half of $F$ do
        $val \longleftarrow \mathsf{PRG}_{blockies}.\mathsf{next}() * 2.3$
        if $val < 1$ then    // happens with prob $\frac{1}{2.3} \approx 43.5\%$
            $p \longleftarrow colorBack$
        else if $1 \leq val < 2$ then    // happens with prob $\frac{1}{2.3} \approx 43.5\%$
            $p \longleftarrow colorFront$
        else    // happens with prob $\frac{0.3}{2.3} \approx 13\%$
            $p \longleftarrow colorSpots$

    Set the right half of $F$ to the mirrored left half of $F$ return $F$
```

Algorithm 2: PRG-based visual hash function

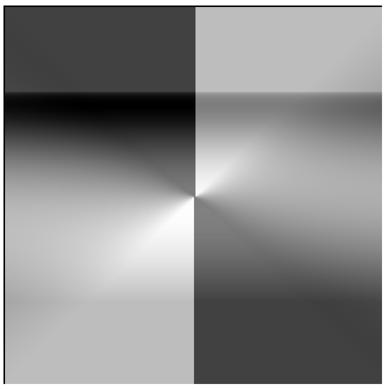

Figure 7: RandomArt visual hash.

6.3.2 RandomArt

[Ref] introduces another way of generating visual hashes using randomness: the idea is again to seed a PRG with the hash, then to use recursion to generate a random function to apply to each pixel. To visualise this, we construct an example. We choose a depth of 4, meaning we have 4 level of nesting. The following function was generated :

```
f(x,y) = SUB(SIN(SQRT(DIV(y,0.56))),SIN(ADD(DIV(y,x),SQRT(-0.35))))
```

In mathematical notation :

$$f(x,y) = \sin\left(\sqrt{\frac{y}{0.56}}\right) - \sin\left(\frac{y}{x} + \sqrt{-0.35}\right)$$

Note that each function maps the interval $[-1,1]$ to itself, using the necessary pre-processing if needed, hence the square root of the negative value will not cause any error.

This function is then evaluated at each pixel. Figure 7 shows the resulting fingerprint. In the true scheme, we repeat the process for each color canal.

The issue with this scheme is that the time needed to generate the random function grows exponentially with depth. Also, similarly to the Blockies scheme, the security of RandomHash relies entirely on the security of the PRG. Also, because of the choice of primitives and their properties, different functions can produce the exact same image; for instance

```
ADD(MULT(x,y), MULT(0.5, x)) == MULT(x, ADD(y, 0.5))
```

because $xy + 0.5x = x(y + 0.5)$ for all x, y by distributivity of addition. However, [ref] gives us another tool: the way they evaluated if their hashes were recognisable by humans. Indeed, they give without proof two criteria to assess the quality of their image:

- The fingerprints should have a high compression factor, as low compression is a sign of a chaotic image.
- The fingerprints' frequency spectra should be concentrated around low frequences, as high frequences typically indicate noise.

The second criterion is what inspired us in the design of our scheme.

7 Our new scheme

RandomHash seems to work well, but the function generation algorithm complexity grows exponentially with depth, which can incur high computing time. So we design a new scheme with the following idea: instead of generating a random function and hoping that its frequency spectrum is centered around low frequences, we generate a frequency spectrum centered around low frequences and apply a 2-dimensional inverse Fourier transform to create the image.

7.1 2D Discrete Fourier Transform

The 2D Discrete Fourier Transform (2DDFT) is defined as follows:

$$\mathcal{F}(u,v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-i2\pi(ux/M + vy/N)}$$

and the 2D Inverse Fourier Transform (2DIDFT):

$$f(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} \mathcal{F}(u,v) e^{i2\pi(ux/M + vy/N)}$$

where M and N are the dimensions of the input image, and $f(x, y)$ denotes the value of the pixel at position (x, y). Although $\mathcal{F}(u, v)$ is defined for any $u, v \in \mathbb{Z}$, since $\mathcal{F}(u + M, v) = \mathcal{F}(u, v + N) = \mathcal{F}(u, v)$, we restrict the input domain to $[0, M - 1]$ for u and $[0, N - 1]$ for v (similarly to how we would sometimes restrict the input domain to $[0, 2\pi)$ for trigonometric functions).

What we would like to achieve is to map a hash to an array of coefficients in the frequency domain, and then apply the inverse 2D Transform on this frequency spectrum to generate a fingerprint.

We see that $\mathcal{F}(u, v)$ is a complex number, meaning each element in the output space of $\mathcal{F}$ requires 2 coefficients to store (modulus and phase). However, there is a symmetry in the output space: indeed, we have

$$\begin{aligned}\mathcal{F}(-u,-v) &= \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-i2\pi(-ux/M - vy/N)} \\ &= \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{i2\pi(ux/M + vy/N)} \\ &= \mathcal{F}(u,v)^*\end{aligned}$$

where $(\cdot)^*$ denotes the complex conjugate. That means we actually have to sample $MN/2$ complex numbers.

Figure X shows a graphical representation of how we generate our spectrum. The darker squares represent a number with a greater module. We now detail the possibilities for the computation of the ω_i.

7.2 Randomness

While having a scheme relying entirely on a PRG is not desirable [WHY?], we may use randomness when generating the ω_i in order to achieve intermediate pre-image resistance. In particular, we tried randomizing any combination of the ω_i moduli and phases. Table 1 shows the computation of the ω_i, where d_i is the evaluation of a distance function at the coordinates of the pixel corresponding to b_i. The exact formula for d_i is described [LATER].

	Deterministic Phase	Random Phase
Deterministic module	$d_i\omega_i = (0.5 + 0.5b_i)///1/// \exp(b_i \cdot \pi i)$	$\omega_i = d_i \cdot b_i \exp(rand() \cdot 2\pi i)$
Random module	$\omega_i = d_i \cdot rand() \cdot \exp(b_i \cdot \pi i)$	$\omega_i = d_i \cdot rand() \cdot \exp(rand() \cdot 2\pi i)$

Table 1: Caption

Name	Formula
`MANHATTAN`	$\frac{1}{\lvert x\rvert+\lvert y\rvert}$
`SQUARE`	$\frac{1}{x^2+y^2}$
`CUBIC`	$\frac{1}{\lvert x\rvert^3+\lvert y\rvert^3}$
`EUCLIDEAN`	$\frac{1}{\sqrt{x^2+y^2}}$
`CUBIC`	$\frac{1}{x^3+y^3}$
`MULT`	$\frac{1}{\lvert xy\rvert}$
`XY`	$\frac{\lvert xy\rvert}{x^2+y^2}$
`BELL`	$e^{-(x^2+y^2)/3}$
`SIGMOID`	if $\frac{1-2\max(\lvert x\rvert,\lvert y\rvert)}{(1-2t)(1+e^{-0.01\lvert xy\rvert})}$

Table 2: Caption

7.3 Distance function

We want our ω_i to have high modules around the axes. That being said, we have to decide exactly which function must the modules follow. We tried several options, detailed in table 2.

7.4 (Temporary) Fourier visual hash scheme