Hash Visualization: a New Technique to improve Real-World Security

Adrian Perrig
Adrian_Perrig@cs.cmu.edu

Dawn Song
Dawn_Song@cs.cmu.edu

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Phone: (412) 268 3052
Fax: (412) 268 5576

Abstract

Current security systems suffer from the fact that they fail to account for human factors. This paper considers two human limitations: First, people are slow and unreliable when comparing meaningless strings; and second, people have difficulties in remembering strong passwords or PINs. We identify two applications where these human factors negatively affect security: Validation of root keys in public-key infrastructures, and user authentication. Our approach to improve the security of these systems is to use hash visualization, a technique which replaces meaningless strings with structured images. We examine the requirements of such a system and propose the prototypical solution *Random Art*. We also show how to apply hash visualization to improve the real-world security of root key validation and user authentication.
Keywords: Human factors in security, hash visualization, user authentication through image recognition, root key validation.

1 Introduction

Although research in security has made tremendous progress over the past years, most security systems still suffer from the fact that they neglect human limitations in the real world. In this paper, we analyze two human limitations: difficulties people have with remembering strong passwords and personal identification numbers (PIN)[1], and second, with comparing meaningless strings[2]. These human factors negatively affect many security systems, including the security of root key validation and user authentication. The problem in root key validation is that people need to compare meaningless key fingerprints, which are strings of 32 hexadecimal digits. It is a known fact in psychology that people are slow and unreliable at processing or memorizing meaningless strings [11, 8]. Also, in [2] Anderson et al. show that strings can be memorized better if people can associate meaning with them, or if they look familiar. Similarly, the problem in user authentication is that people have difficulties with choosing and memorizing strong passwords. If the passwords are too simple and have meanings, they are easy to remember but vulnerable to attacks which use password cracker programs. If the passwords are more complex and random, they are difficult to remember and hence users have to write them down. In either case, the security of the systems is degraded.

These problems have long been considered as some of the fundamental weaknesses of security systems in the real world, we propose to use images to alleviate them. In the case of root key validation we use hash visualization to generate images from the strings, and the user can simply compare the images instead of the strings. This scheme is based on the fact that humans are very good at identifying geometrical shapes, patterns, and colors, and they can compare two images efficiently, as shown in [7, 15, 13]. In the case of user authentication, we replace the precise recall of a password or PIN with a recognition of a previously seen image. Again, it has been shown that people are extremely efficient at recognizing previously seen images [1, 6].

Researchers have been trying to make cryptographic primitives stronger against attacks. The central point of this paper is to show that human factors have a large impact on the security of a real-world system. Our contribution is to propose the new security primitive *hash visualization*, to establish the necessary requirements, to

[1] By *strong* passwords or strong PINs we mean strings which have no immediate meaning or relationship with the person. Therefore an attacker will have difficulties in guessing it.

[2] We use *meaningless* from the point of view of the user. The hash value, or fingerprint of a public-key certificate has a purpose for the program, but no understandable meaning for the user.

propose *Random Art* as a prototypical solution, and finally, to show how to apply hash visualization to improve the security of root key validation and user authentication. Since *Random Art* is just a prototype of the final solution, we hope with this paper to direct the interest of researchers in image processing, security, and psychology, and cooperation between them in order to find better solutions.

The paper is organized as follows. First we examine the requirements of the ideal hash visualization scheme in section 2. In section 3 we propose a possible solution to satisfy the requirements of the hash visualization. We then give in section 4 two example applications about how to apply the hash visualization scheme to improve the security of systems. We discuss some problems and limitations of this approach in section 5 and finally conclude and list our future work in section 6.

2 Requirements for Hash Visualization Algorithms

We first briefly review the definition and desired properties of usual hash functions. We then discuss the properties that hash visualization algorithms should satisfy.

2.1 Review of the requirements for traditional hash functions

This review is based on the Handbook of Applied Cryptography [10].

- A *hash function* is a function h which has, as a minimum, the following two properties:
  1. *Compression*: h maps an input x of arbitrary finite length, to an output $h(x)$ of fixed bit length n.
  2. *Ease of computation*: given h and an input x, $h(x)$ is easy to compute.
- Three most desired properties:
  1. *Preimage resistance*: for any pre-specified output y, it is computationally infeasible to find the input x such that $h(x) = y$.
  2. *2nd-preimage resistance*: given any input x, it is computationally infeasible to find an input x' such that $h(x') = h(x)$.
  3. *Collision resistance*: it is computationally infeasible to find any two distinct inputs x, x' which hash to the same output, $h(x) = h(x')$.
- A *one-way hash function* is a hash function h with two additional properties: pre-image resistance and 2nd-preimage resistance. A *collision resistant hash function* is a hash function h with the additional property of collision resistance.

2.2 Requirements for hash visualization algorithms

Definition 1 *A hash visualization algorithm(HVA) is a function h_I which has, as a minimum, the following two properties:*

1. *Image-generation: h_I maps an input x of arbitrary finite length, to an output image $h_I(x)$ of fixed size.*
2. *Ease of computation: given h and an input x, $h_I(x)$ is easy to compute.*

In order for HVAs to be useful for secure applications, we illustrate a variety of desired properties for HVAs. A HVA that is used in a particular application will only need to satisfy a subset of the properties. We will give several examples of these applications and their usage of the HVAs in the later section.

Near-one-way property

We define two images I_1 and I_2 to be *near*, denoted as $I_1 \simeq I_2$, if the two images are perceptually indistinguishable.

1. *Near preimage resistance*: for any pre-specified output image y, it is computationally infeasible to find the input x such that $h_I(x) \simeq y$.
2. *Near 2nd-preimage resistance*: given any input x, it's computationally infeasible to find x' such that $h_I(x') \simeq h_I(x)$.
3. *Near collision resistance*: it is computationally infeasible to find any two distinct inputs x, x' which hash to the same output, $h_I(x) \simeq h_I(x')$.

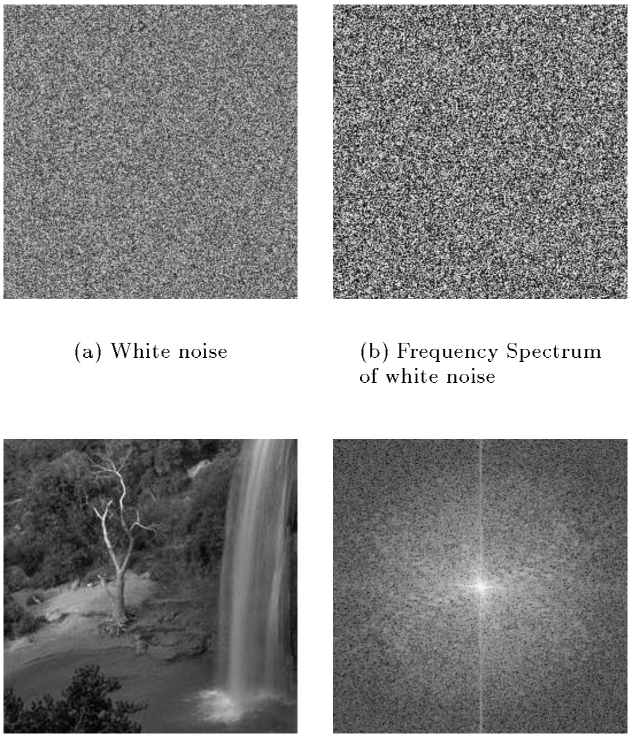
It is difficult to devise an algorithm which can judge automatically whether two images are near since that depends on the person comparing the images. But in general, we can find some similarity-metric function $\delta : I \times I \rightarrow \mathcal{R}$ and a threshold β such that if $\delta(I_1, I_2) \geq \beta$, then the two images I_1 and I_2 are not near. Finding a good function for δ is an active area of research in image retrieval and is not in the scope of this paper.

Regularity property

Humans are good at identifying geometric objects (such as circles, rectangles, triangles, and lines), and shapes in general. We call images, which contain mostly recognizable shapes, *regular images*. If an image is not regular, i.e. does not contain identifiable objects or patterns, or is too chaotic (such as white noise), it is difficult for humans to compare or recall it.

We suggest two ways for testing the regularity of an image automatically.

1. We can use a compression algorithm to compress the image. If the image is chaotic, such as white noise, the compression factor will be very small since almost every pixel is random. Therefore we can show that an image is regular if the compression factor is above a certain threshold.



(a) White noise

(b) Frequency Spectrum of white noise

(c) Photograph

(d) Frequency spectrum of photo

Figure 1: White noise and photograph

2. Non-regular images tend to have wide frequency spectra. Noisy images contain a high percentage of the energy in high frequencies. Hence we can transform an image to the Fourier domain and compute the magnitude spectrum. If the magnitude spectrum does not have too much energy in the high frequencies, $\sum_{f > f_{thresh}} |F(f)| <$ const, then the image is regular.

To illustrate how to use energy in the magnitude spectrum of the Fourier transform in order to decide regularity, we show in figure 1 white noise along with the Fourier transform.

Minimum complexity property

Since the image might be presented in many different ways, i.e., printed in a newspaper, displayed on a color LCD display, or on a TV screen, the result of comparing two images needs to be robust with respect to resolution and color changes: if two inputs x_1 and x_2 are different, then the two outputs $h_I(x_1)$ and $h_I(x_2)$ should not be near with any resolution and color configuration that could occur in the secure system; similarly, if two inputs x_1 and x_2 are equal, then the outputs $h_I(x_1)$ and $h_I(x_2)$ should be near with any resolution and color configuration that could occur in the secure system.

An immediate implication of this property is that an image can not be too simplistic in shapes and patterns, or rely on subtle color differences. Just like for to the regularity property, we could use compression or the frequency spectrum to detect images that are simplistic. For example, compressing an image which has all pixels set to a unique color, should result in a very short file. Also, the frequency spectrum of such a simplistic picture has all the energy in the lowest frequency components.

3 *Random Art*: A Possible Solution

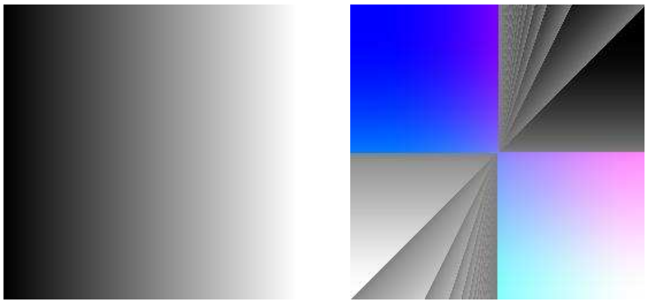
In this section, we propose *Random Art* as a possible solution for the hash visualization algorithm. *Random Art* was developed by Andrej Bauer, and is based on an idea of genetic art by Michael Witbrock and John Mount. Originally *Random Art* was conceived for automatic generation of artistic images. A brief overview and demonstration of *Random Art* can be found at [4].

The basic idea is to use a binary string s as a seed for a random number generator. The randomness is used to construct a random expression which describes a function generating the image—mapping each image pixel to a color value. The pixel coordinates range continuously from -1 to 1, in both x and y dimensions. The image resolution defines the sampling rate of the continuous image. For example, to generate a 100×100 image, we sample the function at 10000 locations.

Random Art is an algorithm such that given a bit-string as input, it will generate a function $\mathcal{F} : [-1,1]^2 \to [-1,1]^3$, which defines an image. The bit-string input is used as a seed for the pseudo-random number generator, and the function is constructed by choosing rules from a grammar depending on the value of the pseudo-random number generator. The function $\mathcal{F}$ maps each pixel (x, y) to a RGB value (r,g,b) which is a triple of intensities for the red, green and blue values, respectively. For example, the expression $\mathcal{F}(x, y) = (x, x, x)$ produces a horizontal gray grade, as shown in figure 2(a). A more complicated example is the following expression, which is shown in figure 2(b).

$$\textsf{if } xy > 0 \textsf{ then } (x, y, 1) \textsf{ else } (\textsf{fmod}(x, y), \textsf{fmod}(x, y), \textsf{fmod}(x, y)), \tag{3.1}$$

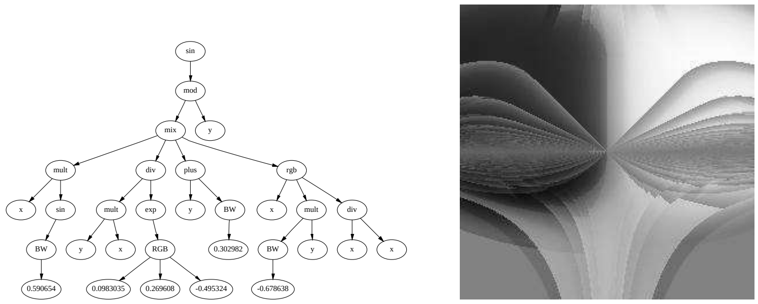
The function $\mathcal{F}$ can also be seen as an expression tree, which is generated using a *grammar* G and a *depth parameter* d, which specifies the minimum depth of the expression tree that is generated. The grammar G defines the structure of the expression trees. It is a version of a context-free grammar, in which alternatives are labeled with probabilities. In addition, it is assumed that if the first alternative in the rule is followed repeatedly, a terminal clause is reached. This condition is needed when the algorithm needs to terminate the generation of a branch.



(a) Image for expression (x, x, x)

(b) Image for expression (3.1) on page 3

Figure 2: Examples of images and corresponding expressions.



(a) Sample *Random Art* expression tree

(b) Generated image

Figure 3: *Random Art* expression tree and the corresponding image

For illustration, consider the following simple grammar:

$$E ::= (C, C, C)^{(1)}$$
$$A ::= \langle \text{random number} \in [-1,1] \rangle^{(\frac{1}{3})} \mid x^{(\frac{1}{3})} \mid y^{(\frac{1}{3})}$$
$$C ::= A^{(\frac{1}{4})} \mid \text{add}(C, C)^{(\frac{3}{8})} \mid \text{mult}(C, C)^{(\frac{3}{8})}$$

The numbers in supscripts are the probabilities with which alternatives are chosen by the algorithm. There are three rules in this simple grammar. The rule E specifies that an expression is a triple of compound expression C. The rule C says that every compound expression C is an atomic expression A with probability $\frac{1}{4}$, or either the function add or mult applied to two compound expressions, with probabilities $\frac{3}{8}$ for each function. An atomic expression A is either a constant, which is generated as a pseudorandom floating point number, or one of the coordinates x or y. All functions appearing in the *Random Art* algorithm are scaled so that they map the interval $[-1, 1]$ to the interval $[-1, 1]$. This condition ensures that all randomly generated expression trees are valid. For example, the scaling for the add function is achieved by defining $\text{add}(x, y) = (x + y)/2$.

The grammar used in the *Random Art* implementation is too large to be shown in this paper. Other functions included are: sin, cos, exp, square root, division, mix. The function $\text{mix}(a, b, c, d)$ is a function which blends expressions c and d depending on the parameters a and b. We show an example of an expression tree of depth 5 in figure 3, along with the corresponding image. For the other images in this paper, we used a depth of 12.

Pseudo-code for the *Random Art* algorithm is shown in Figure 4. The function rnd() used in the algorithm returns a random number in the range $[0, 1)$. The purpose of the 'while' statement in step 5 is to make sure that the expressions do not grow too fast with respect to depth d.

algorithm RandomArt(G, i, d)
input: grammar $G = [r_1, \dots, r_n]$
initial rule i
depth d
output: expression E
begin
(1) Suppose $r_i = [(a_1, p_1), \dots, (a_k, p_k)]$.
(2) If $d \leq 0$ then let $a = a_1$ and goto step (4).
(3) Let a be one of $(a_1, \dots, a_k)$, picking a_i with probability p_i.
(4) If a is a terminal rule let $E = a$ and go to step (6).
(5) Suppose $a = f(r_{i_1}, \dots, r_{i_m})$ where m is the arity of f.
While $d \geq 0$ and $\text{rnd}() \leq 0.5$ do $d := d - 1$.
For each $j = 1, \dots, m$ let $E_j = \text{RandomArt}(G, i_j, d - 1)$.
Let $E = f(E_1, \dots, E_m)$.
(6) Return E.
end

Figure 4: Algorithm *Random Art*

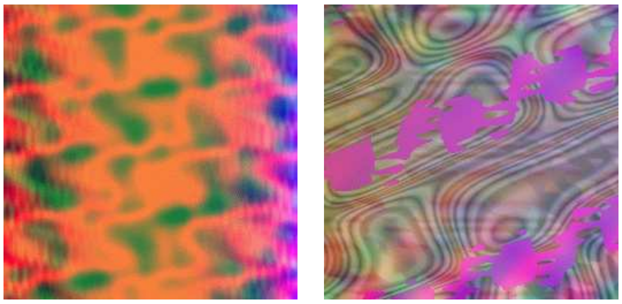
4 Application

In this section, we show how to use hash visualization to improve the real-world security of root key validation and user authentication.

4.1 Validation of root keys using images

In root key validation, a user verifies that a certain locally stored root key really was issued by the corresponding certificate authority (CA). Since the user does not trust data downloaded from the network, the reference fingerprint needs to be passed over another channel, for example printed in a newspaper like the New York Times. Since the reference fingerprint is not in a digital format, the user needs to perform the comparison with the local root key fingerprint manually. This is where security problems due to human factors appear.

We demonstrate our concerns with a brief scenario. A user has just downloaded Netscape Navigator, which comes with a series of top-level root keys. Unfortunately, Netscape has a misleading "verify" button in the window displaying the list of root keys, because "verify" only checks the integrity of the certificate and the dates. Netscape has no way of asserting that the shown key really is the one corresponding to that particular CA.



(a) F_1 (b) F_2

Figure 5: F_1 and F_2 visualized

The only way to verify the key is to obtain the key fingerprint through another channel than the Internet, for example from a newspaper. If we select the "edit" button in the root key window, we can see the following information for the "Verisign Class 1 Primary CA":
Serial Number: *32:50:33:CF:50:D1:56:F3:5C:81:AD:65:5C:4F:C8:25*
Certificate Fingerprint: *51:86:E8:4E:46:D7:B5:4E:29:D2:35:F4:41:89:5F:20*
We call this fingerprint F_1. In the New York Times, we might find the following fingerprint for the same key:
`Fingerprint: 0x5186e81fbcb1c371b51810db5fdcf620`.
We refer to this fingerprint as F_2. A security-conscious user would go through the trouble of validating all 36 root keys that came with Netscape by comparing all the reference fingerprints to the local fingerprints, while many users will most likely not perform all the necessary checks, or only compare the initial or final digits. To compute a public key which will match the 8 initial hexadecimal digits of the fingerprint, it only takes $2^{31} = 2147483648$ trials on average, which is feasible on today's computers. Other users might not understand the importance of verifying the authenticity of the locally stored root key and avoid the validation. Hence, these human limitations greatly degrade the security of the systems. We propose to use Hash Visualization to generate a *visual fingerprint* from the binary fingerprint. When using *Random Art* to generate the visual fingerprints of the two fingerprints F_1 and F_2 listed above, we get the images shown in figure 5.

The visual fingerprints generated by *Random Art* are clearly easier to compare than the hexadecimal representation. Another advantage of this system is that people can remember structured images and recognize them later. Therefore, the user can possibly remember the image representing the fingerprint and perform the validation later. For example, Verisign could display their reference visual fingerprints in an advertisement on TV. At a later point in time, users can display the visual fingerprint on their trusted system and check whether they can recognize the image. Another application of the same idea is the validation of data or software downloaded from the Internet. The scenario is that a business traveler uses the computer that comes with his hotel room to read e-mail. The e-mail program could be a Java applet, downloaded from the Internet, such as the Pachyderm mail reader [14]. The user trusts the hotel computer, but how can he or she know that the Pachyderm applet is correct (i.e. does not contain a Trojan horse)? The obvious solution would be to display a checksum of the Pachyderm applet, which the user could compare with one written down on paper. But again, generating a visual checksum with *Random Art* would be more user-friendly, efficient, and the user would not need to keep a paper with the checksum.

4.2 User Authentication via Image Recognition

Even after years of research in security, authentication schemes based on passwords still have numerous shortcomings [12, 9]. In general, neither simple nor very complex passwords provide the desired security. Shorter, simpler passwords, which are easy to remember, are too easily guessed with a password cracker program and user specific vocabulary. On the other hand, if the password is very complex the user cannot remember it and hence needs to write it on a piece of paper. This again compromises security, since the user might forget, loose, or leave the paper in insecure places. Storing the password in a file might also present a security risk, depending on the computing environment.

Similarly, there is a trade-off related to the number of distinct passwords used. On one hand having many different passwords for different cases of authentication improves the security of a system, but on the other hand users tend to write down the infrequently used passwords, which are usually used for higher security purposes.

Another problem with passwords nowadays is that they are ubiquitous. With the general increase of security awareness, the number of occasions in which a password is required has dramatically increased. Logging onto a computer, accessing a protected spreadsheet file, disabling a secure screen saver, and opening a personalized web site are just a few examples in which a password is required. Since a user can only remember a limited number of passwords, he or she will either write them down, or use similar or even equal passwords for different purposes. Both options have a negative impact on security: writing passwords down increases the chance of compromise, and reusing the same password in different places makes it only as secure as the weakest link.

On the Internet there are sites which offer personalized settings, such as `my.yahoo.com`. These sites require authentication with passwords but often do not use secure communication links. In this way passwords can be easily sniffed off the network, not to mention that a security breach of a site like "My Yahoo!" would compromise a very large number of systems, simply because people use the same passwords on many different systems. Similar considerations apply to PINs, which are frequently used as a method of authentication at ATM's.

The problems presented in this section are common. In the first place, our motive is to draw attention to them, and to stress that even theoretically secure schemes might be insecure in practice because they ignore human factors. Since people cannot remember strong passwords in general, we propose to replace the precise memorization and recall of the password or PIN with a recognition of a previously seen image, with the potential of alleviating some of the problems mentioned above.

Instead of having a user memorize a password, he or she is presented with a small number of images, the *image portfolio*, which he or she must memorize for recognition. The portfolio is shown to the user in a safe environment where it can be ensured that no other person can see the images.

When the user wants to authenticate, he or she is presented with a set of images. Some of the images are chosen from the user's image portfolio and others are generated randomly. The user must correctly identify all the images from the portfolio.

Suppose the portfolio contains n images and that for authentication the system shows $m \geq n$ images. This gives $\binom{m}{n} = \frac{n!}{(n-m)!m!}$ combinations. A credit card PIN code is usually four digits long, which gives 10,000 possible combinations. To match this, we would have to use $n = 5$ and $m = 20$ which gives $\binom{20}{5} = 15504$ combinations.

A disadvantage of current ATM authentication schemes is that PIN codes can be observed from a distance [3] by various ways of acquiring the typing sequence of the key pad. Since the images in our scheme are presented in random order, an observer would gain no knowledge knowing which keys are typed, assuming that the images can only be seen by the person right in front of the ATM. A problem of displaying random images along with the ones in the portfolio is that a criminal could try to log in once for another person, remembering all the presented images. Later, the criminal would make a second attempt, picking the intersection of the presented images, which would correspond to the portfolio. Such attacks need to be taken into consideration during system design.

A combination of a traditional password scheme and image authentication system might give another opportunity to improve the current problems. The key observation for this approach is that people remember the password *approximately*, but not exactly. The system could generate an image for the password which is typed in so far, and the user would then recognize the image corresponding to the correct password and pass that string to the password checking function. Another idea is to use a fixed database of real photographs, instead of *Random Art*, and letting users choose the pictures in their portfolio. This approach can take the advantage that people are extremely good at pointing out which images (or faces) they have seen [1, 6] previously.

5 Analysis and Discussion

In this section we evaluate *Random Art* using the requirements listed in section 2. We first evaluate the geometry and regularity properties, followed by a discussion of how to assess its quality as a hash visualization algorithm.

Geometry and regularity requirements

Similar to the difficulty of formally proving the hash visualization properties, proving that all the images generated by *Random Art* are regular, is hard. However, in practice we can limit the depth of the expression tree, which also limits the complexity of the image. Another factor limiting high complexity is that every function has the same domain and range, which is the interval $[-1, 1]$. Hence, there are no problems with functions approaching infinity, with a subsequent sin function, resulting in a very high frequency signal. In practice, we use 12 for the depth of the expression tree, which has so far resulted in regular images.

As we have shown earlier, we can use the Fourier spectrum to detect irregular (or noisy) images. In figure 6 we show two *Random Art* images with their corresponding Fourier spectrum. Our first observation is that the resulting Fourier spectra are favorable. They resemble the spectrum of a real image (as shown in figure 1(d)), as the energy is concentrated in the low-frequency components. However, image 6(a) is much noisier than image 6(c), as the energy spectrum has more energy in the high frequency components, which can be observed from the corresponding Fourier transforms.

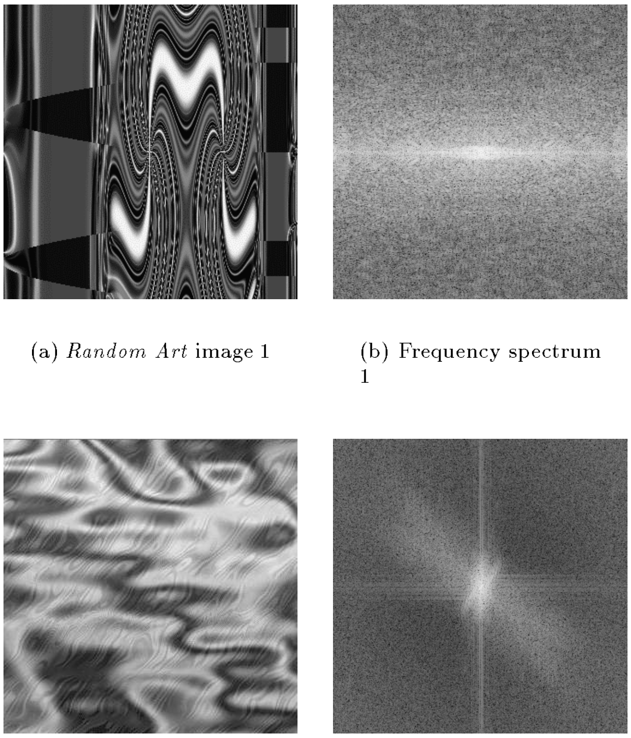
Another issue is to ensure that the resulting images are not too simplistic. We have discussed two ways how to detect simplistic images. One method is compression, where we reject images which compress too well, and the other method is again the Fourier transform, where we can infer simplistic images if all the energy is in the lowest frequency components only.

To get an estimate of how many images in practice are regular or simplistic, we generated 100 images and inspected them manually. It turned out that all images were regular and only 2 were simplistic. This shows that we can generate regular and minimally complex images by detecting and rejecting infrequent outliers.

Hash visualization requirements

For the security of the hash visualization, *Random Art* needs to satisfy the near-one-way requirements listed in section 2.2. We can achieve the hash function one-way properties by hashing the input string with a cryptographically secure hash function, such as SHA-1 [10], to seed a cryptographically secure random number generator, such as Blum-Blum-Shub [5].[3] Hence, we can achieve the pre-image-resistance property. But the difficulty is to achieve collision-free, due to the properties of the image

[3] The original *Random Art* version used in this paper does not use this scheme, but it would be a simple addition.



(a) *Random Art* image 1 (b) Frequency spectrum 1

(c) *Random Art* image 2 (d) Frequency spectrum 2

Figure 6: *Random Art* images with frequency spectrum

generation. First, it is easy to see that many different trees generate identical images. As an example we mention the commutative property of addition or multiplication, where the image remains invariant when swapping sub-trees of those operators. In addition, certain constructs can "destroy" randomness: As an example we consider a construct such as If x < 0.999 then A else B. Since $x \in [0,1]$, the condition is always true if the image is smaller than 1000x1000 pixels, hence the subtree in the else case is never evaluated. In addition, the hash visualization properties take the human factor into account and therefore two images I_1 and I_2 collide if they are near ($I_1 \simeq I_2$). Due to these issues, we could not formally prove that the images generated by *Random Art* satisfy the hash visualization requirements.

Instead, we propose to perform experiments to empirically estimate the difficulty to generate an image collision. With the assumption that all images produced by *Random Art* are equally likely[4], our approach is to pick random seeds and count how many perceptually different images can be generated. To perform the actual counting, we make use of a statistical estimation method, based on the birthday paradox. The birthday paradox basically expresses that if we draw random samples out of a uniform distribution with N distinct values, we start seeing duplicates after approximately $\sqrt{N}$ drawings. The name birthday paradox comes from the fact that we expect with a 50% probability, to have two people with the same birthday as soon as we have more than 24 people.

We can apply this technique to estimate the total number of images in the following way. First, we assume that every image is equally likely. We can then compute the probability that a certain number of collisions were encountered, given the total number of different images N, the number of images generated n, and the number of collisions m. An upper bound for the probability is: $P(N,n,m) \le \binom{n-1}{m}\frac{N!}{N^{n-m-1}(N-n+m+1)!}(\frac{n-m}{N})^m$. But since the user study will reveal for each sample s_i ($1 \le i \le n$) whether it is new or a collision, we can compute the precise probability:

$$\text{Initially,} \quad r = n$$

$$P(N,n,m) = \prod_{1 \le i \le n} \begin{cases} \frac{r}{N} & \text{if } s_i \text{ new}, r := r-1 \\ \frac{N-r}{N} & \text{else } s_i \text{ is a collision} \end{cases}$$

Since only n and m are known from the user study, we can compute N through a maximum-likelihood estimation: which value of N maximizes the probability? It is generally known as a good rule of thumb that squaring the number of samples after the first collision, is a good estimate for N.

Notice that *Random Art* is only a prototypical solution, and hence, might not be the final answer for hash visualization.

6 Conclusion and Future Work

Current security schemes fail to be secure in the real world, because they do not account for human factors. We show how human limitations degrade the security: the difficulty of people to compare or memorize meaningless strings or numbers.

By analyzing two real-world security problems, we show that we could improve their security by taking human factors into account in the system design. We propose to overcome human limitations by replacing strings by structured images.

The two security schemes we analyze are the validation of root keys, and user authentication. The current system to verify the validity of a root key is that users compare the fingerprint of the root key on their computer, with a reference fingerprint distributed over another channel, for example printed in the New York Times. Since this comparison bears many problems, we propose to transform the root key into an image. In this setting, a user needs to compare two images to verify the validity; one in the newspaper and the other on the computer monitor.

In user authentication, people have problems memorizing numbers or passwords. Therefore we propose to replace authentication through string memorization by authentication through image recognition, with the assumption that image recognition is easier than exact string

[4] This does not really hold for *Random Art*, but it gives an estimation of the lower bound.

recall. Our authentication procedure works in the following way. Every user knows a small number of images, the image portfolio. In the authentication process, the user is presented with a number of images, and he or she marks the ones that are from the portfolio. This scheme has additional advantages over other authentication schemes: due to the structure of the images, they can hardly be written down or "explained" to another person.

Since the results presented in this paper are our early findings, there is a lot of work to be done to deploy these methods in reality. First, we need to strengthen the *Random Art* algorithm for this application, in the directions we have pointed out. We then need to evaluate in a user study, how many perceptually different images can be generated. We also need to analyze how people react to the images, and to verify how easy they are to remember for different people. Other directions we are thinking about is to generate a recognizable image, such as a landscape or a city view. The rationale is to bring meaning to the image, which might help with long-term recognition.

7 Acknowledgments

We thank Andrej Bauer for developing the *Random Art* system, and Michael Witbrock and John Mount for the original idea of computer generated art. We are especially grateful to Andrej Bauer for helping us to write section 3. Further we would like to thank everybody who has encouraged us to publish this work, in particular Doug Tygar.

References

[1] Private discussion with Stuart Card, March 1999.

[2] John R. Anderson and Christian Lebiere. *The Atomic Components of Thought.* Lawrence Erlbaum Associates, Inc., 1998.

[3] Ross J. Anderson. Why Cryptosystems Fail. *Communications of the ACM*, 37(11):32–40, November 1994.

[4] Andrej Bauer. Gallery of random art. WWW at http://www.cs.cmu.edu/~andrej/art/, 1998.

[5] L. Blum, M. Blum, and M. Shub. A Simple Unpredictable Pseudo-Random Number Generator. *SIAM Journal on Computing*, 15(2):364–383, 1986.

[6] Kenneth R. Boff, Lloyd Kaufman, and James P. Thomas. *Handbook of Perception and Human Performance.* John Wiley and Sons, 1986.

[7] R. M. Boynton and D. E. Boss. The effect of background luminance and contrast upon visual search performance. *Illuminating Engineering*, 66:173–186, 1971.

[8] Stuart K. Card, Thomas P. Moran, and Allen Newell. The model human processor. In Kenneth R. Boff, Lloyd Kaufman, and James P. Thomas, editors, *Handbook of Perception and Human Performance*, chapter 45. John Wiley and Sons, 1986.

[9] Simson Garfinkel and Gene Spafford. *Practical Unix and Internet Security.* O'Reilly and Associates, 1996.

[10] Alfred J. Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[11] G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1956.

[12] Peter G. Neumann. *Computer Related Risks.* The ACM press, Addison Wesley, 1995.

[13] R. E. Reynolds, R. M. White Jr., and R. L. Hilgendorf. Detection and recognition of colored signal lights. *Human Factors*, 14:227–236, 1972.

[14] Compaq SRC. Pachyderm. WWW at http://www.research.digital.com/SRC/pachyderm, 1998.

[15] L. G. Williams. The effect of target specification on objects fixated during visual search. *Perception and Psychophysics*, 1:315–318, 1966.