On the Security and Usability of Segment-based Visual Cryptographic Authentication Protocols

Tianhao Wang, Huangyi Ge, Omar Chowdhury, Hemanta K. Maji, Ninghui Li
Department of Computer Science
Purdue University
West Lafayette, IN, USA
{tianhaowang, geh, ochowdhu}@purdue.edu, {hmaji, ninghui}@cs.purdue.edu

ABSTRACT

Visual cryptography has been applied to design human computable authentication protocols. In such a protocol, the user and the server share a secret key in the form of an image printed on a transparent medium, which the user superimposes on server-generated image challenges, and visually decodes a response code from the image. An example of such protocols is `PassWindow`, an award-winning commercial product. We study the security and usability of *segment-based visual cryptographic authentication protocols* (`SVAPs`), which include `PassWindow` as a particular case. In an `SVAP`, the images consist of segments and are thus structured. Our overall findings are negative. We introduce two attacks that together can break all `SVAPs` we considered in the paper. Moreover, our attacks exploit fundamental weaknesses of `SVAPs` that appear difficult to fix. We have also evaluated the usability of different `SVAPs` and found that the protocol that offers the best security has the poorest usability.

Keywords

Visual Cryptography; User Authentication; Attack

1. INTRODUCTION

Authentication protocols allow a user to identify herself to a centralized server while ensuring that no adversary, without possession of the secret, can impersonate her. *Human-computable authentication protocols* rely on human's cognitive capability (and, often memory) to carry out the authentication process. These protocols do not rely on the trustworthiness of the user's computing device and hence hold the promise of providing moderately secure and usable second-factor authentication mechanisms. However, most human-computable authentication protocols are either tough to use or very insecure.

The concept of visual cryptography (VC) [26, 27] offers the potential of designing such human-computable, second-factor authentication protocols. VC is a more human intelligible principle of cryptographic protocol design that involves no end-user cryptographic computations and has, subsequently, inspired a wide array of cryptographic applications [30, 20, 5, 14]. A typical VC-based human-computable authentication protocol may proceed as follows. The user and the server share a secret key in the form of an image printed on a transparent key card, *e.g.*, a special credit card with a transparent portion. One authentication session consists of one or more rounds. In each round, an image of challenge frame is shown to the user on a display (*e.g.*, a phone or a computer); then the user overlays her key card on top of the challenge frame to obtain a new superimposed image, interprets the image visually and replies back with a response.

Since VC-based protocols use an image card, it can store a high-entropy key that the user does not need to remember; this opens up the possibility of designing a secure and usable scheme. A prominent representative example of such protocols is `PassWindow` [28, 4], a commercial product available from a startup company, which won the Wall Street Journal Asian Innovation Awards [33]. The design of `PassWindow` relies on the visual decoding of digits represented in 7-segment LED-style and is significantly more user-friendly than other human-computable cryptographic authentication proposals in the research literature.

In this paper, we study the security and usability of *segment-based visual cryptographic authentication protocols* (`SVAPs`), which includes `PassWindow` as a particular case. In `SVAPs`, the images are *segment-based and structured.* More specifically, each frame of image consists of segments such that a group of adjacent segments can be used to display a symbol from some pre-defined set, *e.g.*, digits or letters. We call each such group which can display symbols *a position.*

The security expectation from `SVAPs` is that an eavesdropping attacker (*e.g.*, a malware on the client device) who intercepts the challenge-response pairs belonging to multiple authentication sessions cannot impersonate the user. **Our overall findings are negative**. We introduce two classes of attacks that together can break all `SVAPs` we consider. While some designs (such as having the response computed from multiple displayed digits) can slightly improve security, they hurt usability. Furthermore, our attacks are applicable so long as the images are segment-based and symbol-based.

Contributions. To conduct a systematic study of `SVAPs`, we introduce a framework and a security definition for them. Our security definition is inspired by [25]: An attack algorithm (ℓ, p)-breaks an `SVAP` if it can successfully respond to a *new* challenge with probability p on average after observing a transcript of ℓ successful authentication sessions.

We generalize `PassWindow` to the class of *uni-symbol SVAPs*,

CCS'16, October 24-28, 2016, Vienna, Austria
© 2016 ACM. ISBN 978-1-4503-4139-4/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2976749.2978417

in which each challenge frame can yield at most one symbol (*e.g.*, a digit for `PassWindow`). The security of uni-symbol `SVAPs` relies on the assumption that an adversary is uncertain which position in the frame displays a response symbol. However, through experiences of breaking and fixing `PassWindow` and similar protocols, we gained the following key insight: *Knowing that only one symbol $s \in \Sigma$ is displayed in a frame yields the definite information that no other symbol in Σ is displayed at any position of the frame.* This information can be exploited to reduce the plausible key universe without any expensive searching, and applies to all uni-symbol `SVAPs`.

We develop `Search`, an attack targeting uni-symbol `SVAPs`. `Search` hinges on the following three key ideas. (1) Given a transcript of challenge-response pairs, we can use the above insight to eliminate keys that are inconsistent with the observed transcript. (2) We can perform a bounded search to exploit non-definite information leaked through challenge-response pairs. After the key universe is reduced significantly by exploiting definite information, such a search becomes highly effective. (3) It is not necessary to recover the key fully for the purpose of computing a correct response to challenges. We find that, while `PassWindow` has been found to offer strong security in previous studies [28, 1], and such analysis results have been accepted at face value [11], `Search` is extremely effective against our reconstruction of `PassWindow`, as well as other uni-symbol `SVAPs` we have developed. To summarize, uni-symbol `SVAPs`' fundamental weakness is that each challenge frame encodes at most one symbol.

To overcome this fundamental weakness of uni-symbol `SVAP`, we design *multi-symbol SVAPs* where each challenge frame displays more than one symbol, and the user applies a transformation on the symbols to obtain the response of that frame. Given a frame that encodes two digits d_1 and d_2, we considered two ways to generate the response code: (a) $(d_1 + d_2) \bmod 10$ (*hashing based double-digit SVAP*, in short, `HDD`) and (b) either d_1 or d_2 is acceptable (*either of double-digit SVAP*, in short, `EDD`). We also investigated a generalization of `HDD` to three digits which we identify as *hashing based triple digit SVAP* (`HTD`). `EDD` and `HTD` leaks no definite information, and `Search` is totally ineffective against them.

We introduce the `SolveLP` attack that works for both uni- and multi- symbol `SVAPs`. This attack encodes the impersonation attack as a constraint satisfaction problem, where each variable encodes whether the key contains a particular pattern in a given position. However, off-the-shelf solvers are found to be ineffective at solving the resulting constraints. We thus developed an algorithm for solving them, exploiting the linear programming relaxation technique. That is, instead of assigning binary values to the variables, we consider a relaxation that assigns fractional values to them, and interpret a larger value as more likely to be 1 (cf., the randomized rounding technique [29]). We further apply the iterative multiplicative update technique to solve the resulting linear programming problem. Our experimental evaluation shows that all the newly introduced multi-symbol `SVAPs` are susceptible to this attack, although they do offer higher security than uni-symbol `SVAPs`.

Our security analysis exposes a fundamental limitation of `SVAPs`, that is, one can attain a compact representation of the partial knowledge about the key based on each position independently. `SolveLP` takes advantage of the compact representation by treating whether a specific pattern appears in a given position of the key as a variable to be solved. Any authentication protocol whose human-computable transformation for deriving the response code from the displayed symbols is expressible as constraints is vulnerable to this attack.

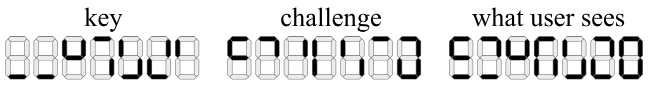


Figure 1: An example of key, challenge and combined frames for an `SVAP` authentication round.

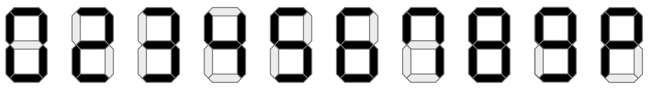


Figure 2: Visual representation of `PassWindow`'s Σ.

Finally, we also carried out a user study on Amazon Mechanical Turk service to evaluate the usability of some of the concrete `SVAPs` we have analyzed for security. Some representative notable findings of our usability analysis are as follows. (1) `EDD` performs the best based on completion time and accuracy. However, `EDD` is very insecure against `SolveLP`. (2) The computation required for `HDD` and `HTD` imposes significant cognitive overhead for the users, and `HTD`, which is the most secure protocol, has significantly worse usability than other protocols. (3) `HDD` offers stronger security than `PassWindow` with similar usability. (4) For `EDD`, conventional wisdom indicates that the chance of the left digit getting picked in higher, au contraire, our study demonstrates that the probability of the user picking the left or the right digit is roughly equal.

Roadmap. The paper is organized as follows. We present our framework, the adversarial model we consider, and the security definition for `SVAPs` in Section 2. Section 3 describes the concrete uni- and multi-symbol `SVAPs` we analyze in this paper. In Sections 4 and 5, we present the `Search` and `SolveLP` attack strategies, respectively. Sections 6 and 7 present the experimental results concerning the `SVAP` security and usability, respectively. We discuss related work in Section 8 and conclude with Section 9.

2. A FRAMEWORK FOR SVAPS

In visual cryptographic authentication protocols, the user and the server share a secret key in the form of an image printed on a transparent key card, which the user superimposes on server-generated challenge images, interprets the resulting image visually, and replies back with appropriate responses. In this paper, we focus on Segment-based Visual Authentication Protocols (in short, `SVAPs`), where each frame of (key or challenge) image consists of segments and is structured so that a group of adjacent segments can encode a symbol. See Figure 1 for an example of the visual key/secret, challenge, and the result of superimposing the key on top of the challenge.

In this section, we first give a general framework for `SVAPs`, then give the adversarial model and the security definition for `SVAPs`. We use `PassWindow` as an example to illustrate the framework.

2.1 The Framework

`SVAP` **Primitives.** Each `SVAP` is based on a Visual Primi-

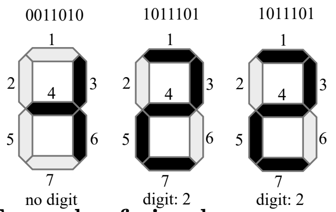


Figure 3: Example of visual component of an SVAP

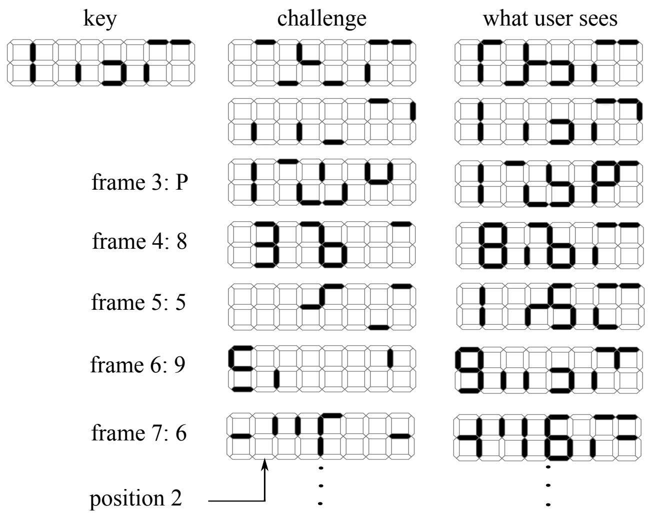


Figure 4: An example key and challenge-response. The response should be the sequence 8596, encoded in frames 4, 5, 6, 7.

tive, which is specified by a tuple $\langle \Sigma, \mathsf{image}, \mathsf{code} \rangle$. Σ gives the set of all symbols used in the SVAP; it can be a set of digits, a set of letters, or some other set of symbols. The function image maps a bitstring to a segment-based image; each segment is turned on (or opaque) if and only if its corresponding bit is '1'. For example, Figure 3 shows how a 21-bit string is visualized as an image with three 7-segment groups. The function code maps a bitstring to a multisets over Σ; e.g., code of the bitstring in Figure 3 yields $\{2, 2\}$.

In SVAPs, the images are created by mapping each bit to a segment; and a group of adjacent segments may display a symbol. We call each such group where a symbol can be displayed *a position*. Multiple positions may overlap with each other. The number of positions is considered a security parameter of an SVAP. Note that overlaying one transparent image on top of the other corresponds to the bit-by-bit 'or' operation of the bitstrings.

In PassWindow, $\Sigma = \{P, 0, 2, 3, \ldots, 9\}$, and their visual representation is given in Figure 2. The letter "P" is used to warn users about the upcoming symbols. The digit "1" is not used because it is shown when just 2 out of the 7 segments are on, and this can leak a significant amount of information and also imposes restrictions on the set of valid challenges and keys.

SVAPs. An SVAP $\mathbf{A}_{\mathsf{V}}$ is a tuple $\langle \mathbf{P}_{\mathsf{V}}, \alpha, \lambda, \mathsf{keyGen}, \mathsf{challengeGen}, \mathsf{response} \rangle$ in which $\mathbf{P}_{\mathsf{V}}$ states the SVAP primitive the protocol is based on, α denotes the number of rounds, *i.e.*, the number of challenge frames that are shown to the user during an authentication session, λ denotes the security parameter, and the other components are explained below.

The function $\mathsf{keyGen}\colon 1^\lambda \to \{0,1\}^{p(\lambda)}$ takes as input the security parameter λ and generates a secret key x of length $p(\lambda)$. For PassWindow, the security parameter λ specifies the number of positions that a digit can be shown. Moreover, two adjacent positions share a common vertical line. Thus, a frame of λ positions consist of $p(\lambda) = 5\lambda + 2$ segments. Figure 4 gives us an example of PassWindow with $\lambda = 8$.

The function challengeGen takes as input a key x, and returns a sequence of α challenges frames $\mathbf{y} = \langle y_1, y_2, \ldots, y_\alpha \rangle$. Challenges are represented in bitstrings.

The function response defines what are considered as valid responses when given a key x and a challenge $\mathbf{y}$. Any response $\mathbf{r}$ in the set of valid responses will be accepted. In Figure 4, the response function will return the response $\mathbf{r} = 8596$.

SVAP Authentication Process. An SVAP is used in the following way. There are two stakeholders, namely, the server (denoted by **Server**) and the user who wants to authenticate herself to **Server** (denoted by **User**). The interaction proceeds as follows.

[**Initialization.**]

- **Server** chooses an SVAP.

[**User registration. (Over a secure channel.)**]

- **User** sends name u to **Server**.
- **Server** generates a secret visual key x using the keyGen function, $x \overset{\$}{\leftarrow} \mathsf{keyGen}(1^\lambda)$.
- **Server** stores $\langle u, x \rangle$ and sends the image $\mathsf{image}(x)$ (likely in the form of a transparency) to **User**.

[**User authentication.**]

- **User** sends name u to **Server**.
- **Server** looks up $\langle u, x \rangle$, generates challenge frames $\mathbf{y} \overset{\$}{\leftarrow} \mathsf{challengeGen}(x)$, and sends the following challenge frames $\mathsf{image}(y_1), \ldots, \mathsf{image}(y_\alpha)$ to **User**.
- **User** overlays the key transparency on top of each of the challenge images, identifies the symbols d_i's, computes the response $\mathbf{r}$, and sends $\mathbf{r}$ to **Server**.
- **Server** accepts the user's response and successfully completes the authentication process *if and only if* $\mathbf{r} \in \mathsf{response}(x, \mathbf{y})$.

2.2 Threat Model and Security Definition

We consider an eavesdropping adversary who can eavesdrop on the communications between **User** and **Server**, *i.e.*, recording the challenges, the responses sent by the user, and the result of whether the responses are correct or not. *In this paper, we limit ourselves to an eavesdropping adversary because an eavesdropping adversary is sufficient for breaking the security of the concrete SVAPs we discuss here.* The objective of the attacker is to successfully impersonate a user, by correctly answering the challenges in one session, without possessing the secret key. We adapt the security definition from [25] as follows.

DEFINITION 1 ((ℓ, p)-BREAK). *We say that an algorithm (ℓ, p)-breaks an SVAP if and only if when given ℓ matching challenge-response pairs and an additional challenge, the algorithm's response to the additional challenge is accepted with probability p on average.*

If an SVAP is $(0, p)$ breakable, then an adversary without eavesdropping ability can impersonate a user with probability p on average in one attempt. When p is above some acceptable threshold, this implies that the protocol is extremely insecure.

3. DESIGN OF SVAPS

We now present two classes of concrete SVAPs which we call the *uni-symbol SVAPs* and *multi-symbol SVAPs*. We assume seven-segment LED-style images for displaying digits are used, as in PassWindow.

3.1 Uni-Symbol SVAPs

In this class of protocols, each frame shows at most one symbol. We categorize these protocols along three design dimensions.

3.1.1 Design Dimensions

1. Noisy Frames or Not. In PassWindow, each authentication session consists of $\alpha = 15$ frames; however, only 4 of them displays a digit, and the 4-digit sequence is the response. These 15 frames are displayed in an animation where each frame displays for around 2 seconds. The animated image loops back to the first challenge frame after the last frame is displayed. A frame encoding the symbol P alerts the users that the next 4 frames encode the digits. The other 10 frames do not display any digit and are known as **noisy frames**.

We call this design the NSD (for **Noisy-frame Single Digit**) scheme, and the other design that does not use noisy frames and simply shows 4 digit-encoding frames with the response being the corresponding 4-digit sequence the BSD (for **Basic Single Digit**) scheme. In BSD, the 4 challenge frames can be shown one by one, with each new frame displayed after another digit is entered. This way, users can respond at their pace.

The use of noisy frames aims at increasing the uncertainty for the adversary regarding which frames encode which symbols. It comes at the usability cost of increasing the time it takes to complete an authentication session and is likely to introduce some stress since users need to decode visually under time pressure. In Section 6 we shall show that using noisy frames decreases the level of security since the noisy frames leak additional information.

2. Key and Challenge Generation Algorithms. We consider two kinds of generation algorithms. In the first kind, keys and challenges are generated by randomly deciding whether each segment should be turned on or off. In the next kind, keys and challenges are generated by randomly selecting a 7-segment **pattern** for each position. In both cases, additional checks are needed to ensure that the key does not already encode a digit, and the challenges when overlaid with the key, resulting in acceptable images (e.g., not encoding two digits in one frame). In this paper, unless explicitly mentioned, the protocols use a random pattern-based algorithms.

3. Shared Edges or Separate Columns. In PassWindow, the two adjacent 7-segment positions share a common vertical edge; we call this design *Shared Edges* (SE, see See Figure 4). The alternative design is not to use shared edges (see Figure 1). In this paper, unless explicitly mentioned, the protocols do not use shared edges.

As will be shown in the experiment (Section 6), we will show these three choices' influence on security one by one. Here we describe five concrete schemes. They each differ from its previous one in one dimension. The first two use noisy frames, and the latter three do not.

3.1.2 Protocols

1. [NSD(7%+SE)] Noisy-frame Single Digit randomly displaying 7% Shared Edges. The first protocol we want to evaluate is PassWindow. NSD(7%+SE) mimics PassWindow by using 15 frames with 10 noisy frames. It uses random segment generation algorithms (which produce key and challenge with a density close to the PassWindow examples that are publicly available [4, 1]), noisy frames, and shared edges. Specifically, the generation algorithms are as follows:

keyGen. Segments are turned on randomly with a probability of 25%; further check ensures that no position in the key displays any symbol.

challengeGen. One first chooses a random symbol σ from Σ, then randomly selects a position p in the challenge frame and then turns segments in position p on so that σ is displayed in position p when combined with the key. For the other non-encoding positions, the segments are turned on randomly such that the challenge frame has a density of about 7%. Further sanity checks ensure that no other position encodes a symbol when overlaid with the key.

2. [NSD(20%+SE)] Noisy-frame Single Digit randomly displaying 20% Shared Edges. This protocol is motivated by the observation (from experiments) that challenge frames in NSD(7%+SE) have limited entropy since the position on the challenge frame with the most number of segments turned on is almost certainly the one encoding the digit. To mitigate this, NSD(20%+SE) displays 20% segments on the challenge frame.

3. [BSD(20%+SE)] Basic Single Digit randomly displaying 20% Shared Edges. This protocol differs from NSD(20%+SE) in that it does not use any noisy frame. Therefore $\alpha = 4$.

4. [BSD(20%)] Basic Single Digit randomly displaying 20% non-shared edges. This protocol differs from BSD(20%+SE)in that no shared edge is used.

5. [BSD] Basic Single Digit randomly displaying valid patterns. BSD differs from BSD(20%)in that the generation algorithms are based on randomly choosing a pattern, as follows:

keyGen. A *valid key pattern* is any seven-segment pattern such that it has between 1 and 6 segments that are turned on, and it does not display any digit. For each position of the key, randomly choose a valid key pattern.

challengeGen. A challenge is generated using the following steps: (1) Randomly choose 4 symbols and 4 patterns such that each pattern is compatible with at least 3 of the chosen symbols. A challenge pattern is compatible with a symbol if there exists a valid key pattern that displays the symbol when overlaid with the challenge. This step aims to ensure that even if the symbol is known, there are at least 3 possibilities for its position. (2) Place these 4 patterns randomly in 4 positions of the frame such that only one position displays a symbol when combined with the key. The segments in all other positions of the challenge frame are off. This step is intended to minimize the amount of information leakage, and will be explained later.

3.2 Multi-Symbol SVAPs

Section 6 demonstrates a fundamental weakness of uni-symbol `SVAPs`. The information that a frame displays a particular symbol entails that all other symbols are *not* displayed at any position. To defeat attacks exploiting this insight, we introduce multi-symbol `SVAPs`, where each challenge frame can generate more than one digit.

We give three concrete protocols, they all use the same keyGen as in `BSD`, which is more secure than ones based on random segments.

1. [HDD] Hashing-based Double Digit. In `HDD`, each challenge frame generates two digits d_1 and d_2. The corresponding response is the one's digit of the sum of d_1 and d_2, which can be viewed as a simple *human computable hashing function*: $(d_1 + d_2) \bmod 10$. This scheme makes it more difficult to rule out which digit is not displayed when knowing the response digit. However, because 1 is never displayed, this protocol still leaks some deterministic information. For example, if 9 is the response digit, then 8 cannot be displayed since it would require an 1 to result in 9.
challengeGen. Randomly choose $2-9$ positions in the challenge frame and randomly choose the same number of valid patterns to show in those positions in the following way: (a) every digit in Σ is compatible with at least 2 patterns; (b) when combined with the key only 2 of the challenge positions display digits.

2. [HTD] Hashing-based Triple Digit. `HTD` eliminates deterministic leakage of `HDD` by having each frame display three digits d_1, d_2, and d_3. The response is the one's digit of the sum of d_1, d_2, and d_3, that is: $(d_1 + d_2 + d_3) \bmod 10$. This scheme requires more computation than other schemes, which may lead to worse usability. The challengeGen for `HTD` is similar to `HDD`, except that each frame in `HTD` displays 3 digits instead of 2.

3. [EDD] Either of Double Digit. `EDD` prevents the deterministic leakage of `BSD` without resorting to arithmetic. A challenge frame superimposed by the visual key induces two digits d_1 and d_2; however, instead of responding with $d_1 + d_2 \bmod 10$, one is asked to respond with either d_1 or d_2. This reduces the entropy of each frame by one bit, but when observing one digit being the response code, any other digit could still be displayed.

4. BREAKING UNI-SYMBOL SVAPS

We now present `Search`, an attack against single-symbol `SVAPs`. For ease of description, we assume that each position that can display a symbol which consists of 7 segments. The attack equally applies to other settings. The attack is based on the idea that after observing each successful authentication session, one can eliminate the keys that are incompatible with the transcript.

Definition 2 (Compatibility). *Given an SVAP* $\mathbf{A}_\mathsf{V} = \langle \mathbf{P}_\mathsf{V}, \alpha, \lambda, \mathsf{keyGen}, \mathsf{challengeGen}, \mathsf{response}\rangle$*, and a transcript of* ℓ *successful authentications,* $\langle \mathbf{Y} = \langle \mathbf{y}_1, \ldots, \mathbf{y}_\ell\rangle, \mathbf{R} = \langle \mathbf{r}, \ldots, \mathbf{r}_\ell\rangle\rangle$*, we say that a key* x *is compatible with the transcript* $(\mathbf{Y}, \mathbf{R})$*, if and only if* $\forall i \in \{1, \ldots, \ell\}$*,* $\mathbf{r}_i \in \mathsf{response}(x, \mathbf{y}_i)$*.*

4.1 Key Universe Representation

One straightforward way to exploit compatibility is to explicitly maintain the set of all keys that are compatible with the transcript seen so far. This requires maintaining one bit for each possible key (whether it is compatible with the transcript), and is infeasible as the resulting state size will be linear in the size of the key space (*e.g.*, for `NSD (7%+SE)`, this is approximately $2^{5\lambda+2}$, which is 2^{77} when $\lambda = 15$).

Thus the first major challenge to exploit compatibility is to decide how to represent the knowledge about the set of plausible keys. A natural approach is to maintain for each segment, whether it must be "on", must be "off", or is uncertain. Using 2 bits for each segment, this requires $2 \times (5\lambda + 2)$ bits for `NSD (7%+SE)`, which is 144 bits when $\lambda = 15$. However, this fails to capture information regarding the inter-relationship among segments. Suppose that from a challenge/response pair, one can conclude that one of two segments must be off (since otherwise it can be decoded into a different response); however, when one maintains information about segment independently, one can only record that this information.

Our solution exploits the structured and segmented nature of `SVAPs`. We maintain information regarding each position consisting of 7 segments. For a 7-segment position, there are 128 patterns, and we use one bit for each pattern, indicating whether this pattern is plausible in this position. This requires a total space of 128λ bits, which is 1920 bits when $\lambda = 15$.

More specifically, we maintain the **global plausible pattern set** $\mathbb{K} = \langle \Pi_1, \Pi_2, \ldots, \Pi_\lambda\rangle$, where $\Pi_i : \{0,1\}^7 \to \{0,1\}$ gives the set of compatible patterns in the i'th position. The downside of $\mathbb{K}$, of course, is that such a representation loses inter-position information (*e.g.*, if this position takes this pattern, then another position cannot be that pattern). However, such information is still preserved in the transcript, and can be dynamically exploited by searching, as we explain below.

4.2 Deterministic Reduction of Key Universe

The critical step in breaking an `SVAP` using searching is to reduce the global plausible pattern set $\mathbb{K} = \langle \Pi_1, \Pi_2, \ldots, \Pi_\lambda\rangle$; this requires *definite knowledge* regarding each position. This led us to discover a fundamental weakness of uni-symbol `SVAPs`: Every challenge-response pair leaks *deterministic information.* For example, knowing that a frame encodes the digit 4 means that one knows *for certain* that none of the other digits is encoded in any position, even if one does not know where 4 occurs. Thus, using the challenge pattern at each position, one could reduce the Π_i's.

Because of such information leakage, the random pattern challenge generation algorithm we introduced for BSD uses empty positions in the challenge whenever possible. A few positions must be non-empty, to provide some uncertainty as to where a response digit is displayed. However, each new challenge pattern that appears in a position enables one to eliminate key patterns that when combined with the challenge pattern would display digits not in the response.

4.3 Search Trees

While the deterministic information can help prune the global plausible pattern set $\mathbb{K}$, combining that with searching can help further refine $\mathbb{K}$. The key challenge is to limit the total space consumed by the search trees to be below the available resources.

Conceptually, we construct one search tree for each session. For each session, there are a number of choices. For example, for protocols that use noisy frames, the first choice is which frame is the first frame that encodes a symbol. In addition, for each such frame, there is a choice on which position displays a symbol. In each node n, we store $\mathbb{K}^n$, a set of plausible keys consistent with choices made to reach that node. The root node uses the global plausible pattern set $\mathbb{K}$. Each node starts with the set from its parent, and uses the current choice made to create the current node to eliminate keys that are incompatible with this choice. For example, against `NSD` schemes, once we have made a choice about which one among the $\alpha = 15$ frames displays P, we know which symbol (if any) each frame encodes; this can be used to reduce $\mathbb{K}^n$. If a node has $\mathbb{K}^n = \langle \Pi_1^n, \Pi_2^n, \ldots, \Pi_\lambda^n \rangle$ such that one of its component $\Pi_i^n = \emptyset$, we know the current choice is inconsistent with the transcript, as no key pattern can be used on the i'th position; thus the node can be removed from the tree.

We construct a tree by maintaining a queue of the frontier nodes to be expanded. To avoid exhaust the memory, we will expand a tree only when its frontier consists of no more than N nodes, where N is set to 1000 in our experiments. Whenever we expand a node (*i.e.*, making another guess), we simultaneously add all the children nodes into the search tree. We know that one of the node in the frontier must be the correct one; however, we do not know which one it is.

Let $\mathscr{F}$ be the forest consisting of all trees. For each tree $T \in \mathscr{F}$, we use $\mathsf{leaves}(T)$ to denote all the leaf nodes of T. Whether they are fully expanded or not, we observe that the following equality must hold.

$$\forall_{T \in \mathscr{F}}, \forall_{i \in [1..\lambda]}, \Pi_i = \bigcup_{n \in \mathsf{leaves}(T)} \Pi_i^n \qquad (1)$$

We use the above relation to prune $\mathbb{K}$ as follows. Given a tree T, if Π_i contains a pattern not in any leaf node of T, then this pattern can be removed from Π_i. Furthermore, whenever Π_i is reduced, if another tree's frontier node includes any pattern not in Π_i, then that pattern should be removed. If any node's Π_i becomes empty, then this node should be removed, which may cause $\Pi_{i'}$ for some i' to be updated. This process computes a fixpoint that satisfies Eq. (1).

We also note that two trees can be combined, by having a cartesian product of their sets of leaf nodes. The product of two nodes result in a node with their corresponding Π_i's intersected.

4.4 Combining Reduction with Searching

Our attack strategy is to combine deterministic reduction with searching in an integrated fashion. When given $\ell > 1$ pairs of challenge/response, we first reduce $\mathbb{K}$ using deterministic information, and then create a search tree for each authentication session. We iteratively perform the following until no change is made to any tree or $\mathbb{K}$. (1) Use Eq. (1) to prune $\mathbb{K}$ and the search trees as much as possible. (2) Expand any search tree if it is not fully expanded and it has no more than $N = 1000$ leaf nodes. (3) If there exist two fully expanded trees in $\mathscr{F}$ such that the product of the sizes of their leaf sets is no more than $N' = 10000$, merge the two trees by computing the cartesian product of their sets of leaf nodes. The algorithmic description of search over one session is shown in Algorithm 1.

4.5 Guessing the Response

The attack strategy described so far is about how to reduce $\mathbb{K}$; we now discuss how to respond to a challenge frame. The basic strategy is for each symbol, find whether there exists a plausible key in $\mathbb{K}$ that results in the symbol when overlaid with the challenge frame. For challenges generated by random segment based algorithms, however, a more effective response-guessing strategy is possible. For each challenge frame, order the symbols that are compatible with $\mathbb{K}$ by the number of segments that need to be added to form them, and guess the one that requires the fewest number of additional segments based on decreasing probability. The intuition is that the symbol(s) that can be formed by adding the least number of segments are the most likely responses. It turns out that using this guessing strategy alone, without any searching, one can guess the response for `NSD(7%+SE)` with probability around 0.1. That is, this guessing attack can $(0, 0.1)$-break `NSD(7%+SE)` (as shown in Figure 5(c)). Note that such heuristic does not work for random-pattern based challenge generation algorithms.

4.6 Effectiveness of Noisy Frames

The usage of noisy frames that do not encode any digit might appear to improve security, as it prevents adversary from knowing for certain which frame corresponds to which digit. This, however, is not the case. First, as each session has only 4 digits, even if one does not know which frame encodes which digit, one can still exploit the information that none of the other 5 digits are shown on any of the $\alpha = 15$ frames to reduce the plausible key space. This, by itself, may already leak more information when compared with the case of using just 4 frames as in `BSD`. With noisy frames, for each position, one has more than $15 \times 5 = 75$ challenge-digit pairs to eliminate plausible key patterns. With just 4 frames, one has $4 \times 8 = 32$ pairs. Second, one can search through each of the $\alpha = 15$ possibilities, and compute the set of plausible keys for each possibility, and remove key patterns that are incompatible with any possibility. Note that after one makes the guess, the noisy frames leak additional information because they cannot encode any digit.

5. ATTACKING MULTI-SYMBOL SVAPS

`Search` is not effective against multi-symbol `SVAPs` as they reveal significantly less amount of deterministic information. Without such deterministic information, the key reduction step of `Search` is not effective. Without reducing the plausible key space, using the search trees is unable to make progress. In this section, we present `SolveLP`, a class of attacks that model the problem as a pseudo-boolean satisfaction problem.

5.1 Constraint Formulation

The key is denoted by x, a challenge frame is denoted by y, and the response is denoted by r. We use $x[p]$ and $y[p]$ to denote the key pattern and challenge pattern in position p of x and y, respectively. We use $[\lambda] = [1 \ldots \lambda]$ to denote the set of positions in a challenge frame or key. We use P_K

to denote the set of valid key patterns and P_C to denote the set of valid challenge patterns.

Variables In our pseudo-boolean constraint formulation, the constraints are over the following $0-1$ variables.

$$X_{p,k} = 1 \text{ if and only if } x[p] = k, \text{ for } p \in [\lambda] \text{ and } k \in P_K.$$

There are $\lambda * |P_K|$ such variables. These variables encode the key we want to attack, and are the variables that we want to solve. To make it easier to explain the constraints, we also introduce the following macro variables that can be expressed using $X_{p,k}$'s.

$$Y_{p,c,s}, \text{ where } p \in [\lambda], c \in P_C, \text{ and } s \in \Sigma.$$

$Y_{p,c,s} = 1$ if and only if for the key we want to attack, using c as the challenge pattern in position p, displays the symbol s; it can be expressed as follows:

$$Y_{p,c,s} = \sum_{k \in \mathsf{compatible}(c,s)} X_{p,k} \qquad (2)$$

where $\mathsf{compatible}(c, s)$ is the set of all valid key patterns in P_K that display s when overlaid with c.

Universal Constraints. These constraints are applicable to all protocols. The following requires that each position of the key takes only one key pattern.

$$\forall p \in [\lambda], \sum_{k \in P_K} X_{p,k} = 1 \qquad (3)$$

Note that the above represents λ constraints, one for each position. These constraints are conjuncted with the constraints below for encoding information revealed in each challenge/response pair.

`EDD` **Specific Constraints.** For `EDD` we have the following constraints for each challenge frame y and response r.

- **The challenge y generates exactly two digits:**

$$\sum_{p \in [\lambda]} \sum_{s \in \Sigma} Y_{p,y[p],s} = 2 \qquad (4)$$

  This says that knowing the patterns on each position in the challenge, the total number of digits that are displayed is exactly 2. Recall that $Y_{p,y[p],s}$ should be expanded according to Eq. (2).

- **The challenge y generates r:**

$$\sum_{p \in [\lambda]} Y_{p,y[p],r} \geq 1 \qquad (5)$$

  Note that the ≤ 2 part is implied by Eq.(4).

`HDD` **Specific Constraints.** For `HDD` we have the following two constraints for each challenge frame y and response r. The first of which states that the response corresponding to frame y is r. This constraint is the disjunction of two disjunctions, one corresponding to the case that two different digits r_1 and r_2 are displayed, and the other corresponding to the case that the same digit is displayed twice.

$$\bigvee_{r_1 \neq r_2 \in \Sigma \wedge (r_1+r_2) \bmod 10 = r} \left(\sum_{p \in [\lambda]} (Y_{p,y[p],r_1} + Y_{p,y[p],r_2}) = 2 \right)$$

$$\bigvee \bigvee_{r_1 \in \Sigma \wedge (r_1+r_1) \bmod 10 = r} \left(\left(\sum_{p \in [\lambda]} Y_{p,y[p],r_1} \right) = 2 \right)$$

The second constraint captures the deterministic information revealed by `HDD`, that is, as $1 \notin \Sigma$, a challenge frame cannot encode a digit d such that $(d+1) \bmod 10 = r$. Note this does not apply to `HTD`.

$$\sum_{p \in [\lambda]} \sum_{k \in \mathsf{compatible}(y[p],(r-1) \bmod 10)} X_{p,k} = 0 \qquad (6)$$

`HTD` **Specific Constraints.** The constraints for `HTD` include only the first constraint for `HDD`, and can be obtained by natural extension. We omit the details.

5.2 Linear Programming Relaxation

After obtaining the constraints for multiple challenge-response pairs $\langle (y_1, r_1) \ldots (y_n, r_n) \rangle$, one can attempt to use any off-the-shelf SMT, SAT, or pseudo-boolean solver to solve for the variables of the form $X_{p,k}$. Such an assignment of the $X_{p,k}$ variables will give us one plausible key which is compatible with the given challenge-response pairs (y_i, r_i). However, in experiments we have found that all the constraint solvers we have tried fail to scale for $\lambda \geq 10$.

To meet this challenge, we apply the linear programming relaxation technique to solving the 0-1 constraint satisfaction problem. That is, instead of assigning binary values to the variables, we assign fractional values to them, and interpret a larger value as more likely to be 1. We further apply the idea of multiplicative updates to solve the resulting linear programming problem.

Our usage of the multiplicative update method can be roughly viewed as interpreting a fractional value assigned to a variable $X_{p,k}$ as the probability of $x[p] = k$. Initially, all the probabilities are uniform, *i.e.*, for all p, k, $X_{p,k} = \frac{1}{|P_K|}$. However, the initial values of $X_{p,k}$ may not respect the constraints from challenge-response pairs, which consequently means we have to update the values of $X_{p,k}$. We update the variable values in a multiplicative fashion. For instance, if we have a constraint $X_i + X_j = t$, and currently $X_i + X_j = s$, we update the the values in the following way: $X_i \leftarrow X_i \frac{t}{s}$ and $X_j \leftarrow X_j \frac{t}{s}$. If the constraint is inequality, we will only update the values (taking the RHS of the inequality as the target) when the inequality does not hold. As $X_{p,k}$ can appear in different constraints, updating $X_{p,k}$ values to satisfy one constraint may end up violating another constraint. Hence, we loop through all constraints until the values of the variables converge (if the cumulative change of the variables is below a small threshold, we use 0.01 in experiments), or a loop threshold (100 in experiments) is reached.

5.3 Attacking EDD

To attack `EDD`, one has a set of linear constraints that are logically and'ed together. We thus can readily apply the above multiplicative update methods. When given the challenge frame y, we enumerate all possible response r, encode y and r using assuming r is the response, and just use this summation (LHS of Eq. (5)) as the weight corresponding to r. The response with highest weight is chosen to be the guess.

5.4 Attacking HDD and HTD

For `HDD` and `HTD`, because of the uncertainty of which pairs of digits are displayed, the constraint for each frame consists of a disjunction, which cannot be directly handled by the multiplicative update approach. We now explain our approach for HDD, which can be easily generalized to HTD. To attack HDD, we have to combine guessing which pairs are displayed with solving the system of constraints.

Given a pair $\langle y, r\rangle$, we enumerate through all possible digits $r_1, r_2 \in \Sigma$ such that $r_1 + r_2 = r$, and compute

$$v^y(r_1, r_2) = \max_{i,j\in[\lambda]:\ i\neq j} Y_{i,y[i],r_1} \times Y_{j,y[j],r_2} \times \prod_{l\neq i,j} Y_{l,y[l],\perp},$$

where $Y_{l,y[l],\perp} = \sum_{k\in P_K} X_{l,k} - \sum_{s\in\Sigma} Y_{l,y[l],s}$. This estimates the probability that r_1 and r_2 are displayed. We then use the pair with the highest value as the weight for this frame.

We compute the weight for all frames in the transcript, and start with the frame with highest weight. The two digits that produces this weight are assumed to be shown in this frame, and encoded as a constraint. We then update the variables using all constraints, and guess the next frame.

We do this until we've guessed 2/3 of all frames. We leave 1/3 frames unguessed because it is more probable to make wrong guesses in the last few frames (as guesses are made in the descending order of confidence), and one wrong guess would make the final guessing inaccurate. This 1/3 ratio is our initial heuristic choice. In the experiments, we found that using a ratio of 1/4 performs slightly better, and a ratio of 1/2 is slightly worse. We did not attempt to further optimize this ratio. When the transcript is updated, the above computation will be repeated.

When given the challenge frame, we compute the score for each response r, which is the sum of $v^y(r_1, r_2)$ such that $(r_1 + r_2) \mod 10 = r$, and choose r with the highest score.

6. EXPERIMENTAL EVALUATION

In this section, we report experimental results concerning the security analysis of SVAPs under the two attacks: Search and SolveLP.

6.1 The Search Attack

Experimental setting. All the reported results here are averaged over 10,000 runs. For each run, we generate a key and 20 sessions (100 for HDD) of simulated challenge/response.

Starting from the first session, for each session we do the following (1) generate a challenge/response pair $(\mathbf{y}, \mathbf{r})$; (2) without $\mathbf{r}$, use $\mathbb{K}$, the current global set of plausible keys, to guess a response to $\mathbf{y}$, output whether the guess is correct; (3) compute the set of compatible responses, and output the size of this set; (4) output the size of plausible keys in $\mathbb{K}$; (5) add $(\mathbf{y}, \mathbf{r})$ to the transcript, and update $\mathbb{K}$.

We aggregate the output from step (2) of 10,000 runs into the probabilities of making a correct guess after using the transcript of a certain number of sessions. We use $\log_2 \frac{1}{p}$ when plotting the graphs. This can be viewed estimating the min-entropies of the guesses. We perform steps (3) and (4) to generate data that enable us to better understand the relationship between sizes of plausible keys, numbers of compatible responses, and first-guess success probability.

We attack the protocols presented in Section 3.1, *i.e.*, NSD(7%+SE), NSD(20%+SE), BSD(20%+SE), BSD(20%), and BSD. We also attack HDD (α =3), as a comparison of the effectiveness between Search and SolveLP. The Search attack is completely ineffective against EDD and HTD. We present results for two security parameters, $\lambda = 15$ and $\lambda = 30$. Larger λ values hurt usability.

Effectiveness of the attack. Figure 5 shows the results of applying Search against the above protocols. From Figure 5, we can observe the following results.

First, from Figures 5(b) and 5(c), we can observe that Search can $(4, 0.25)$-break BSD(20%). This means that after 4 sessions, a guess with succeed with probability 0.24. However, at this point, the number of plausible keys in $\mathbb{K}$ is roughly 2^{20}. The same trend can be seen for other protocols as well. This suggests that it is possible to carry out an impersonation attack even though a large amount of uncertainty regarding the key remains.

Second, from Figures 5(a) and 5(c), we can see that except for BSD, the level of security indicated by min-entropy is significantly lower than that indicated by the number of possible responses. This indicates two things. First, the number of possible responses is not always an accurate indicator of security. Second, random segment-based challenge generation algorithms are vulnerable to our guessing attacks based on the number of segments needed to display a digit.

Third, NSD(7%+SE) (*i.e.*, our attempt to duplicate PassWindow) is extremely insecure. Even a stateless adversary, not having access to any response, can succeed in the first guess with probabilities 10.7% and 8.2% for $\lambda = 15$ and 30, respectively. After intercepting only 3 challenge/response pairs, the success probabilities increase to 66.9% and 26.2%, respectively.

Fourth, while NSD(20%+SE) offers a slightly better security than NSD(7%+SE) when the attacker has 0 or 1 challenge/response pair, its security degrades to the same level as NSD(7%+SE) with 2 or more intercepted sessions; this is because by using denser segments on non-encoding positions, it also leaks more information regarding the key in each session.

Fifth, using noisy frames is a bad idea. This is because the noisy frames actually provide more deterministic information such as which digits are not shown. Removing noisy frames improves the level of security significantly.

Sixth, using shared edges between adjacent positions or not has almost no noticeable impact on the level of security.

Seventh, all the uni-symbol SVAP variants we identified are very insecure. For $\lambda = 15$, Search is able to $(8, 0.25)$-break all of them, i.e., after observing 8 challenge-response pairs, an adversary can successfully impersonate a user with probability 0.25. For $\lambda = 30$, Search can $(12, 0.09)$-break all uni-symbol SVAP protocols.

Finally, Search can $(20, 2^{-9.84})$ break HDD with $\lambda = 15$, indicating that HDD may be considered to be acceptable if this is the best attack we have.

6.2 The SolveLP Attack

Experimental setting. The experimental setting is similar to that of the previous attack; however, we report only the min-entropy. For this attack, instead of reporting the cardinality of the set of plausible keys and responses, we only make a guess and thus report only the success probability of the guess. In this experiment, we evaluate the effect of applying SolveLP on EDD, HDD, HTD, and BSD.

Effectiveness of the attack. Figure 6 illustrates the Min-Entropy of our attack against EDD, HDD, and HTD. We present results for security parameters $\lambda = 15$ and $\lambda = 30$. We now highlight some of our findings.

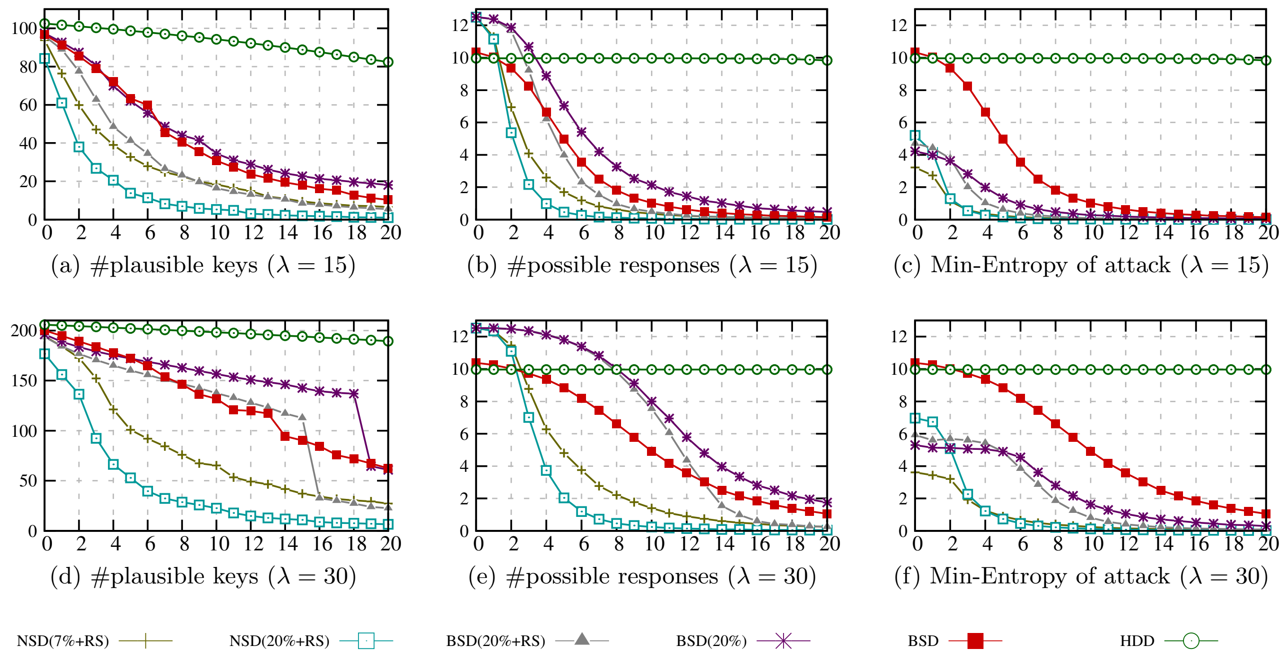


Figure 5: Effect of the Search attack. The X-axes correspond to ℓ, the number of authentication sessions for which the adversary has intercepted the transcripts. The Y-axes in the subfigures 5(a) and 5(d)) correspond to the number of candidate keys. The Y-axes in the subfigures 5(b) and 5(e) correspond to the number of plausible responses. The Y-axes in the subfigures 5(c) and 5(f)) correspond to the inverse of the probability that the first guess succeeds. The values in Y-axes are in Log Scale (Base 2).

First, `EDD` performs the worst under the attack, even worse than `BSD`. This is because there are two, instead of one, correct responses for each frame. `SolveLP` can $(4, 0.25)$-break both `BSD` and `EDD`.

Second, `SolveLP` is more effective than `Search` on both `BSD` and `HDD`. For example, when $\lambda = 15$, `SolveLP` can $(4, 0.25)$-break and $(8, 0.25)$-break `BSD` and `HDD`, respectively, whereas `Search` can $(8, 0.25)$-break and $(8, 2^{-9.97})$-break `BSD` and `HDD`, respectively.

Third, increasing the key length also increases the security of the protocol (see Figure 6). However, increasing the key length to achieve an acceptable level of security yields deployment and usability challenges (*e.g.*, the key card size).

Finally, all the multi-symbol `SVAP` protocols are $(10, 0.25)$-breakable for $\lambda = 15$, and $(20, 0.125)$-breakable when $\lambda = 30$. The results directly exhibit the potency of the `SolveLP` against the multi-symbol `SVAP` protocols and let us draw the conclusion that the concrete `SVAPs` we have considered are insecure in general. Designing protocols that are usable and secure at the same time is a fascinating future research direction.

Security of different protocols. Table 1 presents the levels of security of different schemes in another way. If we view min-entropy of 3 as a point at which we declare a protocol to be broken, Table 1 shows how many sessions a protocol can withstand under an eavesdropping attack.

Efficiency of the attacks. Now we report the running time of our attacks. We measured the wall clock time it takes (*i.e.*, using the Linux `time` utility) to generate and guess 20 sessions using key length 15. All experiments were carried out on a 3.40GHz Intel(R) Core(TM) i7-3770 CPU running GNU/Linux with 16GB RAM.

For `Search`, it takes less than a second to run an attack instance. `SolveLP` takes less than 0.98 minute to run a single instance of `EDD` whereas it takes 1.08 and 1.10 minutes for `HDD` and `HTD`, respectively. Also, we implemented the attacks in python, and we did not aim at optimizing the attack. The reported times here are just representative examples demonstrating the feasibility of the attacks in real life.

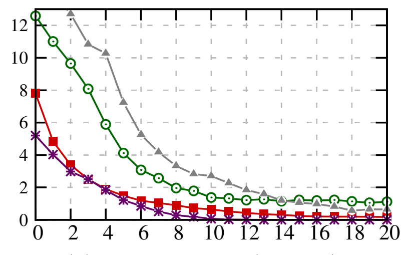
Note that employing `SolveLP` using pseudo-boolean, SAT, or SMT solvers are not feasible in practice. To the best of our knowledge, the best free solvers for our form of constraints are MiniCard [2, 3] (for `EDD`), and MiniSat+ [3] (for `HDD` and `HTD`). Unfortunately, none of the solvers are efficient, especially when the key length is long (*e.g.*, 15). Specifically, for `EDD`, it takes MiniCard around 10 minutes to make a guess for one instance, and much longer for `HDD`.

7. USABILITY EVALUATION

We have conducted a human subject study to evaluate the usability of several `SVAPs`. Our study was vetted by our institution's IRB and was given an exemption on the grounds that it is based on survey procedures, and individual participants cannot be identified from the study.

7.1 Study Design

Protocols Studied. We studied the usability of the following protocols: `NSD(7%+SE)`, `EDD`, `HDD`, and `HTD`. For presentation purposes, we use the alias `ASD` for `NSD(7%+SE)`. For each of the protocols, we displayed the superimposed image (*i.e.*, the key combined with the challenge frame) on a browser, and let the participants respond according to the protocol's authentication requirement. We measured each


(a) Min-Entropy ($\lambda = 15$)

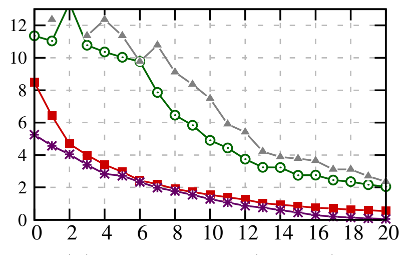

(b) Min-Entropy ($\lambda = 30$)

Figure 6: The X-axes in the curves correspond to ℓ, the number of authentication sessions for which the adversary has intercepted the transcripts. The Y-axes in the subfigures 6(a) and 6(b)) correspond to the inverse of the probability that the first guess succeeds. The values in Y-axes are in Log Scale (Base 2).

	Search					SolveLP			
λ	NSD (20%+SE)	BSD (20%+SE)	BSD (20%)	BSD	HDD	BSD	EDD	HDD	HTD
15	(2.9, 0.8)	(3.7, 1.5)	(3.8, 1.9)	(7.7, 1.4)	(43.6, 4.7)	(3.2, 0.9)	(2.5, 1.1)	(6.7, 1.7)	(11.4, 3.6)
30	(4.3, 1.1)	(7.3, 3.1)	(7.1, 3.6)	(13.1, 2.3)	(83.3, 7.5)	(5.7, 1.1)	(4.5, 1.4)	(13.1, 1.6)	(16.0, 2.9)

Table 1: A value in a cell is of the form (s, k) in which s denotes the average number of sessions for an attack to have min-entropy ≤ 3, i.e., the first guess succeeds with probability $\geq \frac{1}{8}$. The value k denotes the standard deviation of s.

user's accuracy and average completion time of each protocol authentication session. An authentication session of EDD, HDD, and HTD consists of four frames (*i.e.*, $\alpha = 4$), whereas an authentication session of ASD is an animation of fifteen frames among which only four frames display digits which constitute the desired 4-digit response (*i.e.*, $\alpha = 15$). For EDD, HDD, and HTD, we choose α to be 4 so that the response lengths of these protocols are consistent with ASD.

We also noticed the existence of a few 7-segment LED patterns which closely resemble the 7-segment LED patterns representing digits, but are not valid digit-representing patterns. We call these patterns *confusing patterns* (CP). See Figure 7(a) and Figure 7(b) for these CP and their valid counterparts. During the training session, we drew users' attention to those CP to prevent them from getting confused. Meanwhile, we also evaluated the influence of CP on authentication, by considering protocol variants which never display these confusing patterns. We identify the protocol variants which exclude CP with a trailing '-' in their name, *e.g.*, ASD-, EDD-, HDD-, HTD-.

Evaluation Process. We ran our user study through Amazon's Mechanical Turk (MTurk). Each study participant is randomly assigned a specific protocol to use and we require the participant to complete five authentication sessions of that protocol. We evaluated the accuracy rate and the completion time of each participant. Before the evaluation phase, there is a training phase explaining the protocol's authentication requirement.

For ASD and ASD-, the study web page displays an animation of 15 images, with each image lasting two seconds. Participants are can enter the four response digits at any time and then click a button to submit. For the other protocols, the page displays a static image (a single frame), and the participants need to input the response for that frame in order to proceed to the next frame.

The training phase contains a single authentication session with explanations describing the requirement of the task. The authentication session used is similar to the ones used in the real study.

7.2 User Study Result Analysis

For each of the eight protocols (ASD, EDD, HDD, HTD, ASD-, EDD-, HDD-, HTD-), we recruited fifty participants on MTurk. The participants' ages range from 18 to over 50, with about 80% between 23 and 40. Roughly, 80% of the participants hold bachelor's or master's degree. Male participants make up around 60% of all participants. The distributions are similar for all the eight groups.

Figure 8 gives the accuracy and completion time of different protocols. We now highlight some of our interesting findings below.

Accuracy. In terms of accuracy, EDD performs better than ASD ($t = 1.361, p = 0.177$), HDD ($t = 1.925, p = 0.057$), and HTD ($t = 3.096, p = 0.003$) [1]. EDD is followed by ASD, which is slightly better than HDD. HTD is the worst, with accuracy below 80%. This ordering is expected. The differences among these three, however, are not statistically significant.

Completion time. EDD is also the clear winner in terms of completion time (t significant at $p < 10^{-8}$), taking an average of less than 20 seconds per session. EDD is followed by ASD and HDD, each taking an average of around 40 seconds. Given the animated nature of ASD, taking an average of around 40 seconds appears reasonable. If a participant fails to identify a digit, she has to wait for the loop to return to the same frame again. With an animation loop of 30 seconds, users who did not succeed in the first time could take close to 60 seconds. It is a bit surprising that HDD takes as long as ASD. This is due to a combination of the need to

[1] t and p are values used in the t-test to indicate whether a result is statistically significant; t denotes how many standard deviations, and p is the probability; larger t and smaller p mean higher significance.

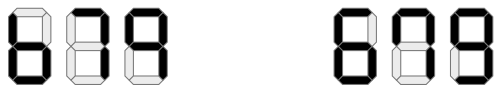


Figure 7: Confusing Patterns and their corresponding Valid Patterns

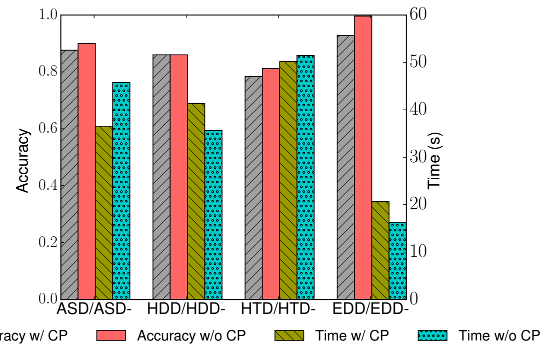


Figure 8: Accuracy and Completion Time of Different Protocols.

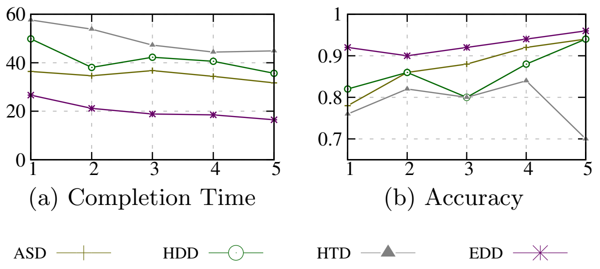


Figure 9: Completion time (in second) and accuracy throughout the five sessions.

recognize two digits and to do the extra arithmetic step. HTD takes the longest, with around 50 seconds per authentication session.

Does practice make perfect? From figure 9(a), we can see that, in most cases, the completion time decreases as the evaluation proceeds. The completion time of the last session is less than the first one in all protocols (t significant at $p < .02$). However, in terms of accuracy (Figure 9(b)), it appears that for EDD and ASD, the accuracy improves. For HDD and HTD, the accuracy rates change more erratically. In particular, in the last session of HTD, the accuracy rate drops by a large value. One possible reason is that participants lost their patience at the last authentication session after performing twenty or so additions of three single-digit numbers.

Overall Assessment. Our results suggest that HDD offers similar usability as ASD, but with higher security. However, the level of security offered by HDD is still far from satisfactory. While HTD offers some security enhancement over HDD, this comes with significant usability cost.

How "Random" is EDD? Among all the participants assigned to EDD or its variant, only a small amount (2%) of participants always choose either the left or the right digit displayed in a frame consistently. Roughly, half of the participants (*i.e.*, 44%) randomly choose between the left and right digits. They choose the left digit roughly with probability 0.4 to 0.6. The distribution of the probability a participant chooses the left digit is similar to a normal distribution. This finding is interesting due to the fact that it contradicts the belief that the probability of the left digit being picked is higher.

Influence of confusing patterns. We evaluated the influence of the confusing patterns (CP) on usability. As shown in Figure 8, removing CP helps increase the users' accuracy for EDD ($t{=}2.982, p{=}0.004$), and at the same time, decrease the completion time for HDD ($t = -1.649, p = 0.1$) and EDD ($t = -2.152, p = 0.03$). For the rest of the protocols, the impact of removing CP in the context of accuracy or completion time is not substantial.

8. RELATED WORK

Human Identification Protocols. The first theoretical foundation of human identification protocol (HIP) dates back to the work by Matsumoto and Imai [23]. However, their scheme was broken by Wang *et al.* [34].

Hopper and Blum [18] proposed to use a NP-hard "*learning parity with noisy*" (LPN) problem. Users can compute inner product of a secret bit-string and a challenge bit-string and uses the result to authenticate herself. Weinshall [35] proposed *Cognitive Authentication Scheme* (CAS) that utilizes human memory and cognition to differentiate two sets of images. However, this scheme is later broken by Golle and Wagner [17]. They used a SAT solver and recovered the secret after intercepting challenge-response pairs of tens of sessions.

Bai *et al.* proposed Predicated-based Authentication Service (PAS) [8] in which the user and the server agree on a predicate. During authentication, the server provides the arguments as challenges and the user applies the predicate to obtain the response. One year later, Li *et al.* [21] took advantage of intersection attack to break this scheme.

Rosener *et al.* proposed GridCode [32] as a layer for obfuscating passcodes, but it was broken by Molloy and Li [24]. They were able to recover the victim's password and additional secret after intercepting a small number of challenge-response pairs.

Human Computable Machine Unbreakable (*HCMU*) hash schemes [10, 9] have been proposed for generating textual passwords [10, 9]. Since they do not use any external memory aid (such as a key card in SVAPs), such schemes are harder to design and to use. For example, the scheme in [10] requires a human user to memorize a random mapping from letters to digits, a random permutation of the digits, and to perform single digit mod 10 additions.

Yan *et al.* [36] studied the inherent complexities of designing authentication mechanisms that do not rely on the underlying computing device. They consider those human-computable authentication protocols that depend on the human users' memory and cognitive capacity. They provide two general classes of attacks that can circumvent these authentication mechanisms. They also provide design principles for leakage-resilient password systems and described a framework for measuring the cognitive overload of a protocol. Their framework and attacks are not directly applicable to SVAPs as SVAPs allow the use of a key card, which can store a high-entropy secret.

Visual Cryptography (VC). Visual cryptography (VC), introduced by Naor and Shamir [26, 27], can be viewed as an instance of human computable cryptography where decoding a ciphertext takes advantage of the visual channel of a human user. In this model, the ciphertext and the secret key are printed images and revealing the plaintext requires superimposing the two printed images (corresponding to the ciphertext and the key) which user can visually observe. Note that individually any of the printed images (ciphertext or key) is indistinguishable from a random noise. Ateniese *et al.* [6] and Lu *et al.* [22] considered VC-based sharing scheme in a graph setting in which a secret image is associated with each edge of the graph. The problem is to generate shares for each vertex of the graph such that when the shares of two distinct vertices v_1 and v_2 are combined, it exposes the secret image corresponding to the edge (v_1, v_2). Ateniese *et al.* [7] then proposed extended VC schemes that allow the shares to be meaningful figures in order to avoid suspicion and censorship. Rijmen and Preneel [31] first proposed a VC scheme to use different colors other than black and white. The same idea was later improved by Hou *et al.* [19]. Chavan *et al.* [15] proposed hierarchical VC (HVC), that can be used to derive more on-demand shares. Abboud *et al.* [5] proposed to combine steganography and VC, but their scheme requires the human computation to be more involved.

Chang and Hu [13] proposed to use VC to distribute shares of a copyright among its stakeholders. Watermark schemes based on VC have also been proposed [20, 16]. VC can also be used to generate voting receipts [14] that enjoy verifiability and coercion resistance.

VC-based authentication. Noar and Pinkas [25] proposed the first framework for VC-based authentication and identification. Their Visual Identification problem is similar to what we call Visual Authentication Mechanism. The security notion we use for SVAPs in our paper is based on the one in [25]. In the scheme they propose in [25], the key consists of a number of squares, each is painted with one of 10 colors. A challenge selects d squares, and the response is the sequence of colors for the d squares, sent in some predefined order. Borchert's note [12] discussed the idea of segment-based VC which can be viewed as a special case of SVAPs. However, in these schemes from each challenge/response pair, an adversary can trivially recover a portion of the key used in the pair.

9. CONCLUSIONS

In this paper, we presented an abstract framework for designing SVAPs. We discussed two classes of protocols: uni-symbol SVAPs and multi-symbol SVAPs. Then, we came up with two attacks Search and SolveLP against them. Finally, we carried out a user study to evaluate the usability of some of the protocols.

Our overall findings are negative; our attacks can break all the protocols we have developed including a commercial protocol. This points to the following two fundamental weaknesses of SVAPs. First, there is not enough entropy in the secret key. Second, SVAPs are highly structured; they do not effectively enlarge the search space of the attack. Therefore, it remains an open problem to design a secure yet usable protocol based on visual cryptography.

10. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1314688. Hemanta K. Maji's research is supported by CNS-1566499.

We thank Matthew Walker for providing some insights and details of the PassWindow scheme. We also thank the anonymous reviewers for their valuable suggestions.

11. REFERENCES

[1] An evaluation of hypothetical attacks against the passwindow authentication method. http://passwindow.com/evaluation_of_hypothetical_attacks_against_passwindow.pdf.

[2] Minicard. https://github.com/liffiton/minicard.

[3] Minisat+. http://minisat.se/MiniSat+.html.

[4] Passwindow. http://www.passwindow.com.

[5] G. Abboud, J. Marean, and R. V. Yampolskiy. Steganography and visual cryptography in computer forensics. In *Systematic Approaches to Digital Forensic Engineering (SADFE), 2010 Fifth IEEE International Workshop on*, pages 25–32. IEEE, 2010.

[6] G. Ateniese, C. Blundo, A. D. Santis, and D. R. Stinson. Visual cryptography for general access structures. *Information and Computation*, 129(2):86 – 106, 1996.

[7] G. Ateniese, C. Blundo, A. D. Santis, and D. R. Stinson. Extended capabilities for visual cryptography. *Theor. Comput. Sci.*, 250(1-2):143–161, 2001.

[8] X. Bai, W. Gu, S. Chellappan, X. Wang, D. Xuan, and B. Ma. Pas: predicate-based authentication services against powerful passive adversaries. In *Computer Security Applications Conference, 2008.*, pages 433–442. IEEE, 2008.

[9] J. Blocki, M. Blum, and A. Datta. Human computable passwords. *CoRR*, abs/1404.0024, 2014. http://arxiv.org/abs/1404.0024.

[10] M. Blum. Mental cryptography and good passwords. http://www.scilogs.com/hlf/mental-cryptography-and-good-passwords/.

[11] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Security and Privacy IEEE Symposium on*, pages 553–567. IEEE, 2012.

[12] B. Borchert. Segment-based visual cryptography. Technical Report WSI-2007-04, Universität Tübingen, 2007.

[13] C.-C. Chang and H. Wu. A copyright protection scheme of images based on visual cryptography. *Imaging science journal*, 49(3):141–150, 2001.

[14] D. Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE security & privacy*, 2(1):38–47, 2004.

[15] P. V. Chavan, D. Atique, D. Malik, et al. Design and implementation of hierarchical visual cryptography with expansion less shares. *arXiv preprint arXiv:1402.2745*, 2014.

[16] W. Chuen-Ching, T. Shen-Chuan, and Y. Chong-Shou. Repeating image watermarking technique by the visual cryptography. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 83(8):1589–1598, 2000.

Algorithm 1 Search

```
Input: Transcript of $\ell$ successful authentication sessions: $\mathbf{Y} = \langle \mathbf{y}_1, \ldots, \mathbf{y}_\ell \rangle$, $\mathbf{R} = \langle \mathbf{r}_1, \ldots, \mathbf{r}_\ell \rangle$.
Output: Global plausible pattern set $\mathbb{K}$.
 1: Initialize $\mathbb{K} = \langle \Pi_1, \Pi_2, \ldots, \Pi_\lambda \rangle$ such that each $\Pi_i$ includes all patterns that can be generated by keyGen
 2: for $j \in [1 \ldots \ell]$, $k \in [1 \ldots \alpha]$, $i \in [1 \ldots \lambda]$ do
 3:     Remove from $\Pi_i$ patterns that when overlaid with $\mathbf{y}_{j,k}$, the $k$'th challenge frame of the $j$'th session, results
 4:     in a symbol that does not appear in $\mathbf{r}_j$, the observed response code for that session.
 5: $F \leftarrow []$
 6: for $j \in [1 \ldots \ell]$ do
 7:     $T \leftarrow []$; $T$.enqueue(newNode($\mathbb{K}, [c_1, c_2, \ldots, c_t]$)); $F$.add($T$)    ▷ $[c_1, c_2, \ldots, c_t]$ are all the choices for $j$'th session
 8: while Making progress do
 9:     for $T \in F$ do                                                       ▷ Expand search trees.
10:         while $T$ not fully expanded AND $T$.size() $\leq 1000$ do
11:             ($\mathbb{K}'$, choices) $\leftarrow T$.dequeue()          ▷ dequeue() returns first node in $T$ that has non-empty choices
12:             $c_h \leftarrow$ choices.dequeue()      ▷ The first element in choices is removed from choices and assigned to $c_h$
13:             for $b \in$ {all possibility of $c_h$} do
14:                 Remove from $\mathbb{K}'$ patterns that are inconsistent with the choice $b$.
15:                 if $\Pi'_i \neq \emptyset$ for each $\Pi'_i \in \mathbb{K}'$ then
16:                     $T$.enqueue(newNode($\mathbb{K}'$, choices)
17:     while there exist two trees $T_i, T_j \in F$ that are fully expanded and $T_i$.size()$\times T_j$.size()$<10000$ do    ▷ Merge two trees.
18:         $F$.remove($T_i$);  $F$.remove($T_j$);  $T \leftarrow T_i \times T_j$;  $F$.add($T$);
19:     for $T \in F$ do                                                       ▷ Fixpoint update.
20:         Remove from $\Pi_i$ any pattern that does not appear in the $\Pi_i$ component of any node in $T$.
21:         for node $n = (\mathbb{K}' = \langle \Pi_1^n, \Pi_2^n, \ldots, \Pi_\lambda^n \rangle,) \in T$ do
22:             Update each $\Pi_i^n$ to remove any pattern not in $\Pi_i$; if $\Pi_i^n = \emptyset$ for some $i$, remove the node from $T$
    return $\mathbb{K}$
```

[17] P. Golle and D. Wagner. Cryptanalysis of a cognitive authentication scheme. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 66–70. IEEE, 2007.

[18] N. J. Hopper and M. Blum. Secure human identification protocols. In *Advances in cryptology—ASIACRYPT*, pages 52–66. Springer, 2001.

[19] Y.-C. Hou. Visual cryptography for color images. *Pattern Recognition*, 36(7):1619–1629, 2003.

[20] Y.-C. Hou and P.-M. Chen. An asymmetric watermarking scheme based on visual cryptography. In *Signal Processing Proceedings, 2000. WCCC-ICSP 2000. 5th International Conference on*, volume 2, pages 992–995. IEEE, 2000.

[21] S. Li, H. J. Asghar, J. Pieprzyk, A.-R. Sadeghi, R. Schmitz, and H. Wang. On the security of pas (predicate-based authentication service). In *Computer Security Applications Conference, 2009.*, pages 209–218. IEEE, 2009.

[22] S. Lu, D. Manchala, and R. Ostrovsky. Visual cryptography on graphs. *Journal of Combinatorial Optimization*, 21(1):47–66, 2009.

[23] T. Matsumoto and H. Imai. Human identification through insecure channel. In *Advances in Cryptology —EUROCRYPT*, pages 409–421. Springer, 1991.

[24] I. Molloy and N. Li. Attack on the gridcode one-time password. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 306–315. ACM, 2011.

[25] M. Naor and B. Pinkas. Visual authentication and identification. In *Advances in Cryptology—CRYPTO*, pages 322–336. Springer, 1997.

[26] M. Naor and A. Shamir. Visual cryptography. In *Advances in Cryptology-EUROCRYPT'94*, pages 1–12. Springer, 1995.

[27] M. Naor and A. Shamir. Visual cryptography ii: Improving the contrast via the cover base. In *Security protocols*, pages 197–202. Springer, 1997.

[28] S. Nettle, S. O'Neil, and P. Lock. Passwindow: A new solution to providing second factor authentication. *VEST Corporation*, 2009.

[29] P. Raghavan and C. D. Tompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, Dec. 1987.

[30] P. Revenkar, A. Anjum, and W. Gandhare. Secure iris authentication using visual cryptography. *arXiv preprint arXiv:1004.1748*, 2010.

[31] V. Rijmen and B. Preneel. Efficient colour visual encryption or shared colors of benetton. *rump session of EUROCRYPT*, 96, 1996.

[32] D. K. Rosener, W. O. Brown, and E. L. Reuss. User authentication system and method, Mar. 31 2008. US Patent App. 12/060,031.

[33] J. WAGSTAFF. A new way to outwit internet fraudsters. Wall Street Journal 11 July. 2010.

[34] C.-H. Wang, T. Hwang, and J.-J. Tsai. On the matsumoto and imai's human identification scheme. In *Advances in Cryptology —EUROCRYPT'95*, pages 382–392. Springer, 1995.

[35] D. Weinshall. Cognitive authentication schemes safe against spyware. In *Security and Privacy, 2006 IEEE Symposium on*, pages 6–pp. IEEE, 2006.

[36] Q. Yan, J. Han, Y. Li, and R. H. Deng. On limitations of designing leakage-resilient password systems: Attacks, principles and usability. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2012.

APPENDIX

A. THE SEARCH ALGORITHM

Algorithm 1 gives the algorithm to reduce $\mathbb{K}$, the set of plausible key patterns, when given a transcript, as described in Sections 4.2, 4.3, and 4.4.