

mapped to codeword  $c_1$ , the location of the value  $x$  is bounded. Clearly, the codeword  $c_1$  contains a lot more information about the location of value  $x$  in FIG. 11B than codeword  $c$  does in FIG. 11A even though the value  $x$  and the codeword selected by the decoding function  $f$  have the same relationship in FIG. 11A and FIG. 11B.

[0097] Returning again to FIG. 2, the security calculator 118 (FIG. 1) calculates a security value for a security parameter based on the sequence of codewords (STEP 240). The security of a visual password is dependent on the codewords that are used to enroll and authenticate a user.

[0098] In one embodiment, the security parameter is entropy. Entropy  $H$  represents the average time to guess the secret pattern. Entropy  $H$  can be calculated with the following equation in which a stochastic variable  $P$  takes on the values  $P_1, P_2, \dots, P_n$  with the probabilities  $p_1, p_2, \dots, p_n$ :

$$H(P) = -\sum_i p_i \log p_i.$$

[0099] Although any logarithm can be used to define entropy, computer security applications customarily use the base 2 logarithm so that entropy can be measured in bits. The base 2 logarithm is used here for that reason.

[0100] In another embodiment, the security parameter is minentropy. Minentropy  $H_m$  represents the chance of guessing the secret pattern in one guess. Minentropy  $H_m$  can be calculated with the following equation in which  $p_{\max} = \max_i p_i$ :

$$H_m(P) = -\log p_{\max}.$$

[0101] The comparator 120 (FIG. 1) compares the security value for the security parameter to the threshold value 122 (STEP 250). In one such embodiment, the threshold value for minentropy is forty to sixty bits. For comparison, six random lower case letters represent 28 bits of minentropy. If the security value does not meet or exceed the threshold value 122, the graphical interface 112 will prompt the user 126 to enter another secret pattern.

[0102] If the security value meets or exceed the threshold value 122, the secret pattern will be accepted. In one such embodiment, the enrollment process of FIG. 2 accepts a secret pattern and saves the sequence of codewords  $\{c_1, c_2, \dots, c_n\}$  produced in STEP 230 as the visual password for use in authenticating the user. In another such embodiment, the process of FIG. 2 accepts a secret pattern and uses the sequence of codewords  $\{c_1, c_2, \dots, c_n\}$  produced in STEP 230 to generate a cryptographic secret, which may be regenerated later from the visual password.

[0103] In some embodiments, the enrollment device prompts a user to train himself to remember the accepted secret pattern. In one such embodiment, the graphical interface 112 will prompt the user to enter a point by displaying the associated image. In another embodiment, the graphical interface displays the original discrete graphical choice during or after the user makes a graphical choice.

[0104] Referring to FIG. 12, in one embodiment, the graphical interface 112 displays a line 1230 between the point 1210 in the secret pattern associated with an image and

the point 1220 on the image that the user selected during the training process. This line 1230 visually shows the user the difference between the two points. In a similar embodiment, the graphical interface 112 displays the line 1230 during the training selection process.

[0105] Referring now to FIG. 13, a flowchart for an enrollment process in accordance with another embodiment of the invention is illustrated. STEPS 1310, 1320, and 1330 of FIG. 13 are similar to STEPS 210, 220, and 230 of FIG. 2. Additionally, an offset 8 between each value  $x$  and the corresponding selected codeword  $c$  is calculated by an offset calculator (STEP 1360). An offset is intended to be used during the authentication process to enable some input values, which would not otherwise do so, to generate the codewords that were generated during the enrollment process. A sequence of offsets  $\{\delta_1, \delta_2, \dots, \delta_n\}$  is thereby generated from the sequence of values and the sequence of codewords. Each offset  $\delta$  is an  $n$ -bit string that expresses the difference between the two  $n$ -bit strings that are a value  $x$  and the corresponding codeword  $c$ . A value  $x$  is likewise equivalent to the corresponding codeword  $c$  and the associated offset  $\delta$ , mathematically expressed as  $x=c+\delta$ . An offset  $\delta$  may be denoted mathematically as  $\delta \in \{0, 1\}^n$  such that  $x=c+\delta$ . In the geometric analogy illustrated in FIG. 9, the offset  $\delta$  between a value  $x$  and the corresponding codeword  $c_3$  is defined as  $(u-170, v+95)$  where  $u$  and  $v$  are the two axes of the  $u$ - $v$  plane.

[0106] In some embodiments of the enrollment process that include calculating an offset  $\delta$ , a codeword  $c$  is selected at random from the set of codewords associated with a decoding function  $f$  for the value  $x$  (STEP 1330). In these embodiments, the decoding function  $f$  is not used during the enrollment process to decode the value into a codeword. Instead, the calculated sequence of offsets (STEP 1360) are used in conjunction with the decoding function  $f$  during the authentication process to bring each input value in the sequence of input values within the correction threshold of the randomly selected codewords.

[0107] FIG. 10 can also be used to illustrate a geometric analogy of an embodiment of the enrollment process that does not involve a decoding function  $f$ . For example, the codeword  $c_2$  may be selected at random from the set of codewords  $C=\{c_1, c_2, c_3, c_4\}$  for the value  $x$ . In such an embodiment, the fact that codeword  $c_2$  is not the codeword closest to the value  $x$  does not matter. Such embodiments ignore the minimum distance of the error-correcting code for the purposes of the enrollment process.

[0108] Referring again to FIG. 13, the generated sequence of codewords is hashed by a hasher (STEP 1370). Hashing effectively conceals the information that is hashed. In one embodiment, the hashing is done with a one-way function, known as a hash function  $h$ , that takes an input and produces an output such that is impractical to figure out what input corresponds to a given output and to find another input that produces the same output. Known hash functions take an arbitrary length input and produce an output of fixed length. This process can be expressed mathematically as  $h: \{0, 1\}^n \rightarrow \{0, 1\}^1$ . In one embodiment, the hasher may accept an  $n$ -bit string representing one or more codewords and produce a longer 1-bit string representing the hash of the codewords. In one embodiment, the hash function produces a binary string with a length 1 of approximately 160 bits.