Journal of Universal Computer Science, vol. 3, no. 3 (1997), 185-196
submitted: 15/2/96, accepted: 8/3/97, appeared: 28/3/97 © Springer Pub. Co.

# On the Weaknesses of Gong's Collisionful Hash Function

S. Bakhtiari
(Centre for Computer Security Research
University of Wollongong, Australia
shahram@cs.uow.eud.au)

R. Safavi-Naini
(Centre for Computer Security Research
University of Wollongong, Australia
rei@cs.uow.eud.au)

J. Pieprzyk
(Centre for Computer Security Research
University of Wollongong, Australia
josef@cs.uow.eud.au)

**Abstract:** This paper presents an attack on Gong's proposed collisionful hash function. The weaknesses of his method are studied and possible solutions are given. Some secure methods that require additional assumptions are also suggested.

**Key Words:** Safety/security in digital systems, Guessing attack, Collisionful hash functions, Message authentication.
**Category:** D.4.6, K.6.5.

## 1 Introduction

*Collision Resistant Hash functions*, or simply *Hash functions*, have been used for producing secure checksums since the 1950's. A hash function maps an arbitrary length message into a fixed length message digest, and can be used for message integrity [Bakhtiari et al. 95a, Damgård 89, Preneel 93]. For this purpose, a sender calculates the message digest of the message and sends it appended to the message. The receiver verifies the checksum by recalculating it from the received message and comparing it with the received checksum (traditional hashing). It is assumed the the probability of finding a collision — two messages that result in the same checksum — is less than a small number.

Before hash functions, encryption algorithms were used to provide authenticity. Using encryption algorithm, one can ensure both confidentiality and authenticity of a message. However, encryption algorithms are not efficient and meanwhile some of them fall into the export restriction. Hence, hash functions are demanded when only authenticity is required.

Ordinary hash functions are not secure against an active intruder who can modify both message and its checksum. Protection against spoofing can be obtained by *Message Authentication Code (MAC)*, in which the checksum depends on both message and a secret key. Berson, Gong, and Lomas [Berson et al. 93] have introduced the new term *Keyed Hash Functions*, where the MAC is constructed from an existing hash function, by adding the key to the hash function inputs. A keyed hash function uses a symmetric key and the checksum can only be calculated and verified by the insiders — people who know the key.

Berson *et al.* [Berson et al. 93] have also introduced *Collisionful Hash Functions*, in which many keys result in the same checksum of a given message, and hence, the probability of determining the correct key, used by the communicants, is reduced.

Gong [Gong 95] has given a construction of collisionful hash functions to be used for software protection. This paper analyzes this construction and shows how an enemy can modify a message and its corresponding checksum. The weaknesses of other variations of his method are also studied and experimental results that support our approach are included.

Gong's method is described in Section 2. Section 3 examines the problems of Gong's hashing scheme and demonstrates how to attack the system. Practical experiments which support our claims (attacks) are presented in Section 4. Two secure methods that require additional assumptions are given in Section 5. Finally, we conclude the paper in Section 6.

## 2 Gong's Collisionful Hash Function

A keyed hash function is a class of one-way and collision resistant hash functions, indexed by a key [Bakhtiari et al. 95b, Berson et al. 93]. The hash value depends on the key, and the computation of the key should be infeasible when a reasonably large number of [message, digest] pairs are available.

Collisionful hash functions provide an additional property which is the possibility of having the same hash value of a given message under several keys. This property reduces the chance of uniquely determining the correct key, for the opponent [Berson et al. 93].

Gong [Gong 95] used polynomial interpolation to construct a collisionful hash function with the collision accessibility property. This property allows a user to choose a set of keys that satisfy a given [message, digest] pair, and is desirable when the key belongs to a distinguishable subset of the key space (eg. meaningful words).

A similar approach is used by Zheng *et al.* [Zheng et al. 93] in construction of *Sibling Intractable Function Family (SIFF)*. It is used for providing secure and efficient access in hierarchical systems and has proven security properties. We show that Gong's construction, which is insecure for protection against modification, can be turned into a SIFF. This results in a proof of the security of that construction when a large password space is used.

### 2.1 Notations and Assumptions

- $A$ and $E$ are the user (Alice) and the intruder (Eve), respectively.
- $M$ is a message (a system binary code) to be authenticated by $A$.
- $\mathcal{P}$ is the set of all possible passwords (publicly known).
- $\mathcal{K}$ is a **small** subset of $\mathcal{P}$, so that an attacker can perform an exhaustive search [Gong 95, Sections 1 and 4] (publicly known).
- $k_1 \in \mathcal{K}$ is $A$'s password.
- $k_2, \ldots, k_n \in \mathcal{P}$, $n \in \mathbb{N}$, are selected password collisions.
- $\Delta = \{k_1, k_2, \ldots, k_n\}$.
- $GF(p)$ is the Galois Field of $p$ elements, where $p$ is a prime.

- $K$ is a random key chosen by Alice from a large space (eg. $GF(p)$, for a large prime $p$).
- $g()$ is a secure keyed hash function, where $g(k, x)$ denotes the hash value of a message $x$ under a key $k$.
- $|\mathcal{X}|$ denotes the size of a set $\mathcal{X}$.
- '$\|$' denotes string concatenation.

In this scheme, Alice intends to protect a binary code or file (referred to as a message). She calculates a checksum, derived from the message and her password, and appends it to the message. She will verify the checksum whenever she wants to use the message. An example of such scheme is to safe-guard executable files against viruses.

It is assumed that $g()$ produces integer hash values and $g(k_i, M) > n$, $\forall k_i \in \Delta$. Furthermore, $g(k_i, M) \neq g(k_j, M)$, $\forall k_i \neq k_j$, where $k_i, k_j \in \Delta$. Note that, $\mathcal{K} \subset \mathcal{P}$ is the set of passwords that are commonly used by the users. In general $|\mathcal{P}|$ may not be small, but $\mathcal{K}$, the set of passwords that are often used by the users (called *poorly chosen passwords*), is usually small, and therefore, weak against dictionary attack (cf. Section 3). It should be emphasized that Gong's construction assumes a small password space that can be exhaustively searched:

> *"... collisionful hash functions are useful in generating integrity checksum from user passwords, which tend to be chosen from relatively small space that can be exhaustively searched."* [Gong 95, Section 4]

## 2.2 Computing the Checksum

Alice ($A$) chooses a random key $K$ and defines $w(x) = K + a_1 \cdot x + \cdots + a_n \cdot x^n \pmod{p}$, where $p$ is a suitable large prime number, and the $n$ coefficients $a_1, \ldots, a_n$ are calculated by solving the following $n$ equations. (Equation '$i$' is $w(g(k_i, M)) = k_i$, and all calculations are performed in $GF(p)$.)

$$\begin{cases} K + a_1 \cdot g(k_1, M) + \cdots + a_n \cdot g(k_1, M)^n = k_1 \\ K + a_1 \cdot g(k_2, M) + \cdots + a_n \cdot g(k_2, M)^n = k_2 \\ \qquad \vdots \qquad\qquad\qquad\qquad\qquad \vdots \\ K + a_1 \cdot g(k_n, M) + \cdots + a_n \cdot g(k_n, M)^n = k_n \end{cases} \tag{1}$$

Using $K$ and $w(x)$, the checksum will be:

$$w_1 \parallel w_2 \parallel \cdots \parallel w_n \parallel g(K, M), \tag{2}$$

where $w_i = w(i)$ is a $(\log_2 p)$-bit number for $i = 1, \ldots, n$. Alice does not need $K$ and $w(x)$, and may forget them after producing the above checksum.

## 2.3 Verifying the Checksum

To verify the checksum, Alice needs to remember only $k_1$ (assuming that $p$ is publicly known). She solves the following $(n + 1)$ equations in $GF(p)$ and finds the $(n+1)$ variables $b_0, b_1, \ldots, b_n$.

$$\begin{cases} b_0 + b_1 \cdot g(k_1, M) + \cdots + b_n \cdot g(k_1, M)^n = k_1 \\ b_0 + \quad b_1 \cdot 1 \quad + \cdots + \quad b_n \cdot 1^n \quad = w_1 \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ b_0 + \quad b_1 \cdot n \quad + \cdots + \quad b_n \cdot n^n \quad = w_n \end{cases} \tag{3}$$

Then, she calculates $g(b_0, M)$ and compares it with $g(K, M)$ in the checksum. In the case of a match, she will accept the checksum as valid.

## 3 Attacking Gong's Method

$E$ exhaustively searches $\mathcal{K}$ (cf. Gong's assumption in Section 2.1) and for each candidate password $k \in \mathcal{K}$ solves Equation 3 in $GF(p)$, by replacing $k_1$ with $k$, and finds $b_0, b_1, \ldots, b_n$. The complexity of solving this equation depends on both $n$ and $|\mathcal{K}|$ $(O(|\mathcal{K}|n^3))$. (We have practically examined our attack for $n = 5$ and $|\mathcal{K}| = 2^{20}$. The probability of our attack increases for larger $n$'s (cf. Section 3.1).) If $g(b_0, M)$ is the same as that in the checksum, she keeps $k$ as an *applicable* password. After exhaustively testing $\mathcal{K}$, Alice will find $m$ applicable passwords $\Gamma = \{k_{r_1}, \ldots, k_{r_m}\}$.

**Theorem 1.** $(\Delta \cap \mathcal{K}) \subseteq \Gamma$*, and therefore,* $m$ *is greater than or equal to the number of passwords chosen from* $\mathcal{K}$*.* (Proof is given in the Appendix.)

In the following, we consider Gong's basic and extended constructions and give our attack in each case. Alternative methods with higher security are also suggested. It is important to notice that our attack is aimed at forging a valid checksum without requiring the specific value of $k_1$.

### 3.1 Attacking the basic scheme ($m \leq n$)

It is not unexpected to have $m \leq n$. When $K, a_1, \ldots, a_n$, and $M$ are given, it is improbable to find a password $k \notin \Delta$ such that $K + a_1 \cdot g(k, M) + \cdots + a_n \cdot g(k, M)^n = k$, because $|\mathcal{K}|$ is usually much smaller than $|GF(p)|$ and there is no guarantee one can find a $K'$ $(\neq K)$ such that $g(K', M) = g(K, M)$. (Note that, $g()$ is not a collisionful hash function. For instance, define $g(k, x) = h(k\|x\|k)$, where $h()$ is a collision resistant hash function.) Furthermore, $k_2, \ldots, k_n$ are selected from $\mathcal{P}$ $(\supseteq \mathcal{K})$, and an exhaustive search on $\mathcal{K}$ might not give all passwords $k_2$ to $k_n$. This decreases $m$, the number of applicable passwords. (Note that, $k_1 \in \mathcal{K}$.)

If $m < n$, the attacker $E$ randomly selects $(n - m)$ passwords $(\notin \Gamma)$ and adds them to $\Gamma$. Using the resulting $n$ $(= m)$ passwords (which include $k_1$), the opponent uses the procedure given in Section 2.2 to calculate the checksum for an arbitrary message $M'$ and a randomly chosen key $K'$. Contrary to Gong's claim that the chance of a successful guess is as most $\frac{1}{n}$ [Gong 95, Page 169], the probability of a successful attack is 1, when $m \leq n$.

As mentioned before, choosing $k_i$'s, $2 \leq i \leq n$, from $\mathcal{P}$ will reduce the number of resulting applicable passwords $(\Gamma)$, which is more desirable in our attack. However, we consider a more secure version of Gong's method, by assuming that $k_i \in \mathcal{K}$, $i = 1, \ldots, n$. With this assumption, Theorem 1 results in:

**Corollary 2.** *If $k_i \in \mathcal{K}$, $i = 1, \ldots, n$, then $\Delta \subseteq \Gamma$, and therefore, $m \geq n$.*

The case $(m = n)$ falls into the basic scheme, which is already considered. Now we examine the case when $(m > n)$.

## 3.2 Attacking the extended scheme $(m > n)$

Gong [Gong 95, Page 170] extends his method by calculating the checksum as,

$$w_1 \parallel w_2 \parallel \cdots \parallel w_n \parallel g(K \bmod q, M), \tag{4}$$

for a suitable $q \in \mathbb{N}$. Employing modular reduction increases $m$, the size of $\Gamma$, if $q < |\mathcal{K}|$. Gong does not disclose the state of $n$ $(= |\Delta|)$, and therefore, we consider two cases in which $n$ is either fixed (always the same $n$ being used) or it is an arbitrary integer (which may vary every time). Note that, for a given message $M$, Alice may fix the number of password collisions $(n)$ and publicize it such that the corresponding hash value will be accepted only if it is verified by $n$ password collisions. This is not discussed in the original paper, however.

If $n$ is not fixed, $E$ can construct a fraudulent checksum by solving the following $m$ equations in $GF(p)$, for $a_1, \ldots, a_m$,

$$\begin{cases} K' + a_1 \cdot g(k_{r_1}, M') + \cdots + a_m \cdot g(k_{r_1}, M')^m = k_{r_1} \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ K' + a_1 \cdot g(k_{r_m}, M') + \cdots + a_m \cdot g(k_{r_m}, M')^m = k_{r_m} \end{cases} \tag{5}$$

where $K'$ is a randomly chosen key and $M'$ is an arbitrary message; and calculating the new checksum as,

$$w(1) \parallel w(2) \parallel \cdots \parallel w(m) \parallel g(K' \bmod q, M'), \tag{6}$$

where $w(x) = K' + a_1 \cdot x + \cdots + a_m \cdot x^m \pmod p$. In this case, when $A$ wants to verify the above checksum for the forged message $M'$, she takes $g(K' \bmod q, M')$ off the checksum and divides the size of the remaining part by $\lceil \log_2 p \rceil$ to find the number of chosen password collisions. She follows the procedure in Section 2.3 which will result in the acceptance of $M'$ as a genuine message.

If $n$ is fixed, the attack will still succeed with significant probability. Let $m = n + 1$. Then $E$ can solve the following $(n+1)$ equations for $(n+1)$ variables $a_0, a_1, \ldots, a_n$,

$$\begin{cases} a_0 + a_1 \cdot g(k_{r_1}, M') + \cdots + a_n \cdot g(k_{r_1}, M')^n = k_{r_1} \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ a_0 + a_1 \cdot g(k_{r_{n+1}}, M') + \cdots + a_n \cdot g(k_{r_{n+1}}, M')^n = k_{r_{n+1}} \end{cases} \tag{7}$$

where $M'$ is an arbitrary message. The valid checksum for $M'$ will be,

$$w(1) \parallel w(2) \parallel \cdots \parallel w(n) \parallel g(a_0 \bmod q, M'), \tag{8}$$

where $w(x) = a_0 + a_1 \cdot x + \cdots + a_n \cdot x^n \pmod p$.

If $m > n+1$, since $\Delta \subseteq \Gamma$ (cf. Corollary 1), $E$ can randomly choose $(n+1)$ passwords $\{k_{t_1}, \ldots, k_{t_{n+1}}\}$ from $\Gamma$ and have $A$'s password $(k_1)$ among them, with the probability of:

$$\Pr[\, k_1 \in \{k_{t_1}, \ldots, k_{t_{n+1}}\}\,] = \frac{\binom{m-1}{n}}{\binom{m}{n+1}} = \frac{n+1}{m},$$

which is a high probability if $m$ is not much larger than $n$.

Now, $E$ can use $\{k_{t_1}, \ldots, k_{t_{n+1}}\}$ to solve Equation 7 for $a_0, \ldots, a_n$, and calculate a valid checksum for an arbitrary message $M'$ (Equation 8). That is, $E$ can generate a valid checksum with the probability of $\frac{n+1}{m}$.

For example for $n = 9$, $A$ should ensure that the number of the applicable passwords $(m)$ will be at least $10^5$ to decrease the probability of attack to $10^{-4}$, which is the probability of guessing a 4-digit number — typically the size of the password used by bank Automatic Teller Machines (ATM). This would be a difficult task due to the fact that $m$ is not controllable — it depends on $|\mathcal{K}|$, $p$, and $q$.

### 3.3 Attack by discarding some of the applicable passwords

Another attack on Gong's method is to reduce the size of $\Gamma$, the set of all applicable passwords, by discarding the inappropriate passwords.

We have already proved that if $n$ is not fixed or $m \leq n+1$, there are attacks in which $E$ succeeds with the probability of 1 (100%). Now, assume $n$ is fixed and $m > n+1$, and denote by $\Lambda = \{K_{k_{r_1}}, \ldots, K_{k_{r_m}}\}$ the collection of the resulting keys that correspond to the passwords in $\Gamma = \{k_{r_1}, \ldots, k_{r_m}\}$ (cf. Section 3). Note that, if at least two password collisions are chosen from $\mathcal{K}$, $\Lambda$ will have some repeated elements. This is true because $K_{k_i} = K_{k_j}$, $\forall k_i, k_j \in \Delta \subseteq \Gamma$. We partition $\Lambda$ into $l$ sub-collections $\Lambda_1, \ldots, \Lambda_l$ corresponding to the distinct values of $\Lambda$. That is, $K_\alpha = K_\beta$, $\forall K_\alpha, K_\beta \in \Lambda_t$, $t = 1, \ldots, l$.

On one hand, it is obvious that there exists a $\Lambda_t$ such that $K_{k_i} \in \Lambda_t$, $\forall k_i \in \Delta$. On the other hand, to derive $K_{k_\alpha} \in \Lambda$ from $k_\alpha \in \Gamma$, we used Equation 3 which maps the small password space $\mathcal{K}$ into the large key space $GF(p)$. Therefore, one cannot expect to come up with many (more than $n$) distinct passwords $(\notin \Delta)$ that are mapped to the same key $K \in \Lambda$. Hence, the above $\Lambda_t$, where $\{K_{k_1}, \ldots, K_{k_n}\} \subseteq \Lambda_t$, can be easily distinguished among the other portions, and the claim is emphasized when $n$, the number of password collisions, is large (cf. the experimental results in Table 2).

Consequently, we can select $\Lambda_t$ as the portion that includes $K_{k_1}, \ldots, K_{k_n}$, and use the techniques in the previous sections to attack the method. In particular, even if $n$ is fixed and $|\Lambda_t| > n+1$, the probability of successful attack is $\frac{n+1}{|\Lambda_t|}$ $(\geq \frac{n+1}{m})$.

To decrease the probability of an attack, Alice should use a collisionful hash function in Equation 1, not a keyed hash function $g()$. Also, if she can find one more key collision set of at least $n$ elements which result in a different key $K'$, she can halve the success probability. However, this is a very difficult task due to the difficulty of solving Equation 3 for $b_1, \ldots, b_n$, and $k_1$, when $b_0$ is a given

| $\lvert\mathcal{K}\rvert$ | $\lvert\Gamma\rvert$ |
|---|---|
| $2^{10}$ | 5 |
| $2^{12}$ | 5 |
| $2^{14}$ | 5 |
| $2^{16}$ | 5 |
| $2^{18}$ | 5 |
| $2^{20}$ | 5 |

Table 1: *Basic scheme, where the checksum is* $w_1 \parallel \cdots \parallel w_n \parallel g(K, M)$ *and* $\Delta \subset \mathcal{K}$. *For* $n = 5$, *the number of resulting applicable passwords* ($\lvert\Gamma\rvert$) *was exactly equal to 5, in all cases. Therefore,* $\Gamma = \Delta$ ($m = n$).

fixed key. In other words, since $g()$ is generally not invertible, it is hard to find $s$ ($\geq n$) key collisions $k_{t_1}, \ldots, k_{t_s}$ that result in a fixed $b_0$ ($\neq K$), using Equation 3.

### 3.4 Attack by using $t$ pairs of [message, checksum]

One restriction to Gong's method is that whenever $k_1$ ($A$'s password) is used to calculate checksums for other messages, the same password collisions should be used. Otherwise, the intruder can guess $k_1$ from the intersection of different password collision sets, with a high probability. Suppose $t$ pairs of [message, checksum] are available and enemy has found the corresponding $t$ sets of acceptable passwords. If the size of the intersection of these sets is less than $n + 2$, the techniques given in Section 3.2 can break the system (100%). Otherwise, this intersection can be partitioned, similar to the way described in the previous section, to select the appropriate set with a high chance. This probability will significantly increase when $t$, the number of the given pairs, increases. In fact, the chance of reducing the number of guessed passwords to $k_1, \ldots, k_n$ will significantly increase.

Also, Alice should be careful when $k_1$ is used for other purposes, since some information about $k_1$ is always leaked from Gong's method. For instance, if $k_1$ is also used for logging into a system, the enemy can use our attack, guess all possible password collisions, and try them one by one until she logs into the system.

## 4 Practical Results of the Attack

Authors have implemented Gong's method and the corresponding attacks on a SUN SPARC station ELC. The experiments completely coincide with the previously mentioned theories and support our claims about the weaknesses of the proposed selectable collisionful hash function.

Table 1 illustrates the results of our attack on the basic scheme. In all cases, the number of password collisions ($n$) was chosen to be 5. It shows that in all cases we could exactly find the five password collisions and forge the checksum based on Section 3.1.

Table 2 is the results of our attack on the extended scheme. Different modulo reductions are examined, where in all cases we could select the exact valid password collisions based on Section 3.3. It is important to notice that Section 3.4

| | | | *Partition of $\Lambda$* | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lvert\mathcal{K}\rvert$ | $q$ | $\lvert\Gamma\rvert$ | $\lvert\Lambda_1\rvert$ | $\lvert\Lambda_2\rvert$ | $\lvert\Lambda_3\rvert$ | $\lvert\Lambda_4\rvert$ | $\lvert\Lambda_5\rvert$ | $\lvert\Lambda_6\rvert$ | $\lvert\Lambda_7\rvert$ | $\lvert\Lambda_8\rvert$ | $\lvert\Lambda_9\rvert$ | $\lvert\Lambda_{10}\rvert$ | $\lvert\Lambda_{11}\rvert$ | $\lvert\Lambda_{12}\rvert$ |
| $2^{10}$ | 127 | 14 | 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | |
| $2^{12}$ | 511 | 16 | 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $2^{14}$ | 2047 | 14 | 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | |
| $2^{16}$ | 8191 | 13 | 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | |
| $2^{18}$ | 32767 | 12 | 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | |
| $2^{20}$ | 131071 | 13 | 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | |
| $2^{10}$ | 511 | 7 | 5 | 1 | 1 | | | | | | | | | |
| $2^{12}$ | 2047 | 8 | 5 | 1 | 1 | 1 | | | | | | | | |
| $2^{14}$ | 8191 | 7 | 5 | 1 | 1 | | | | | | | | | |
| $2^{16}$ | 32767 | 5 | 5 | | | | | | | | | | | |
| $2^{18}$ | 131071 | 8 | 5 | 1 | 1 | 1 | | | | | | | | |
| $2^{20}$ | 524287 | 8 | 5 | 1 | 1 | 1 | | | | | | | | |

Table 2: *Extended scheme, where the checksum is* $w_1 \| w_2 \| \cdots \| w_n \| g(K \bmod q, M)$ *and* $\Delta \subset \mathcal{K}$. *For* $n = 5$, *the number of resulting applicable passwords* ($|\Gamma|$) *was usually larger than 5, but after partitioning* $\Lambda$, *in each case, there was only one partition* ($\Lambda_1$) *with 5 elements* ($\Lambda_1 = \Delta$). *This table is part of our extensive experimental results. Two different modulo reductions* ($q$) *are examined for every instance of* $\mathcal{K}$.

gives even a more powerful attack when several checksums are available. However, we could break the scheme without assuming multiple available checksums.

The results show that with Gong's assumptions (especially the small password space), it is possible to attack his method. In the next section, we present methods which are secure if additional assumptions are met.

## 5 Securing the Method

In this section we show that the security of Gong's method under certain restricting assumptions is related to the security of *Sibling Intractable Function Families (SIFF)* [Zheng et al. 93]. This ensures the security of the scheme for a large password space. However we note that assuming a large password space might not be realistic in practice and hence propose alternative methods that reduce the probability of a successful attack.

### 5.1 Gong's Construction and SIFF

In this section, we want to show that Gong's construction can be turned into SIFF [Zheng et al. 93]. Suppose a message $M$, a randomly chosen key $K \in GF(p)$, and $n$ password collisions $k_1, \ldots, k_n$ are given. Define $h(x) = g(x, M)$, where $g()$ is a secure keyed hash function. We note that $h()$ is one-way, because $g()$ is one-way on both parameters. We further assume that $h()$ is collision resistant. An example of $h()$ which satisfies these assumptions can be obtained if we start from a collision resistant hash function $H()$, and define $g()$ as $g(k, M) = H(k \| M)$. It can be seen that $g(k, M)$ is collision resistant on both parameters and hence $h(x) = g(x, M)$ will be collision resistant.

Now calculate $x_i = h(k_i)$, $i = 1, \ldots, n$ and solve the $n$ equations,

$$\begin{cases} a_1 \cdot x_1 + \cdots + a_n \cdot x_1^n = k_1 - K \\ a_1 \cdot x_2 + \cdots + a_n \cdot x_2^n = k_2 - K \\ \quad\vdots \qquad\qquad\qquad\quad \vdots \\ a_1 \cdot x_n + \cdots + a_n \cdot x_n^n = k_n - K \end{cases}$$

for $a_1, \ldots, a_n$. The above is the rearrangement of Equation 1. Therefore, one may use the same technique, given by Gong, to calculate the checksum of a given message $M$. We claim that if $u()$ is defined as,

$$u(k, y) = k - a_1 \cdot y - \cdots - a_n \cdot y^n,$$

then $(u \circ h)$ defined as $u(k, h(k))$ is an $n$-SIFF, when $h()$ is chosen to be a one-to-one and one-way function family. This is true because of the way we defined $u()$ and $h()$ (cf. [Zheng et al. 93]). Note that $u(k_i, x_i) = K$, for $i = 1, \ldots, n$, and therefore, $u(k_i, h(k_i)) = K$, for $i = 1, \ldots, n$. That is, $(u \circ h)$ will map all keys $k_1, \ldots, k_n$ to the same value $K$. The reader is referred to [Zheng et al. 93] for a more detailed description of SIFF.

The above ensures the security of Gong's construction if $g()$ is properly chosen and, in practice, implies that for a large password space the method resists all possible attacks.

## 5.2 Small Password Space

As noted in section 5.1, the security of Gong's construction can only be guaranteed for large password spaces. Also, the whole idea behind the construction of collisionful hash functions, and in particular Gong's method, is to take advantage of small key space. These assumptions might not be realistic in practical cases. In the following, we propose alternative solutions by relying on more reasonable assumptions which can provide smaller chance of success for an intruder.

1. Suppose $A$ always uses $n$ passwords $k_1, \ldots, k_n$ to calculate the checksums (they can be words chosen from a phrase). She can solve,

   $$\begin{cases} a_0 + a_1 \cdot g(k_1, M) + \cdots + a_{n-1} \cdot g(k_1, M)^{n-1} = k_1 \\ \quad\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ a_0 + a_1 \cdot g(k_n, M) + \cdots + a_{n-1} \cdot g(k_n, M)^{n-1} = k_n \end{cases}$$

   for $a_0, \ldots, a_{n-1}$, and calculate the checksum as $g(a_0, M)$. This will be appended to the message $M$ and can be verified only by solving the above equations.
   An adversary ($E$) should guess $n$ passwords from $\mathcal{K}$ and check whether the resulting $a_0$ satisfies the checksum (there are $\binom{|\mathcal{K}|}{n}$ possible selections). A proper choice of $n$ will prevent $E$ from finding the correct selection which results in the genuine $g(a_0, M)$.
   Moreover, if $g()$ is collisionful on the first input parameter (cf. [Berson et al. 93]), $E$ will not be sure that she has found the right passwords. In fact, $E$ may

find an $a_0'$ such that $g(a_0', M) = g(a_0, M)$, but it does not necessarily result in $g(a_0', M') = g(a_0, M')$, for another message $M'$.
A disadvantage of this method is the difficulty of memorizing $n$ passwords, when $n$ is large. However, we do not need a very large $n$ in this method. One may consider a short sentence or phrase as the password.

2. Let $c$ be the least integer such that $2^c$ computations are infeasible. Further assume a user password has on average $d$ bits of information. (Clearly, $d < c$ and $2^d$ computations are feasible.) The checksum of a given message $M$ is calculated as $h(k_1 \parallel R \parallel h(M))$, where $k_1$ is $A$'s password, $R$ is a randomly chosen $(c-d)$-bit number, and $h()$ is a collision resistant hash function [Bakhtiari et al. 95a, Preneel 93].
To verify the checksum, $A$ exhaustively tests $2^{c-d}$ possible values of $R$ and calculates $h(k_1 \parallel R' \parallel h(M))$ for each candidate $R' \in GF(2^{c-d})$. A match indicates that the checksum is valid, because $h()$ is collision resistant. Since both $k_1$ and $R$, which have in total $d+(c-d) = c$ bits of uncertainty, should be guessed by an enemy to verify the checksum, a random guessing attack is thwarted.
Note that, this verification has a maximum overhead of $2^{c-d}$ computations, but instead, selectable password collisions are not demanded. Furthermore, one may use $h((h(k_1) \bmod 2^b) \parallel R \parallel h(M))$, for a suitable integer $b$ $(\leq d)$, to provide password collisions. In this case, $R$ should be $(c-b)$ bits.
For example, assume $c = 64$, $d = 50$, and $h()$ results in 128-bit digests. $A$ can verify the checksum $h(k_1 \parallel R \parallel h(M))$ by computing $h()$ for at most $2^{14}$ candidate $R$'s. This takes about 2 seconds on a SUN SPARC station ELC, when $h()$ is MD5 [Rivest 92]. Verification time is almost independent of the message length, since $h(M)$ needs to be calculated only once (not $2^{14}$ times).
Disadvantage of this method is the difficulty of finding a constant $c$ which suits all users. In practice, different computing powers result in different values of $c$. Therefore, the largest amount should be chosen, which is not desirable on slow machines, because $2^{c-d}$ computations may become time consuming.

## 6 Conclusion

We showed that Gong's collision-selectable method of providing integrity is not secure, and an attacker with reasonable computing power can forge a checksum of an arbitrary message (or binary code). Assuming extra properties for the underlying hash function, it is possible to prove the security of Gong's construction under all attacks, when the password space is large.

Finally we have proposed alternative methods that require additional assumptions and meanwhile provide higher security (smaller chance of success for the enemy).

## A Proof of the Theorem

For any $k_i \in (\Delta \cap \mathcal{K})$, Equation 3 becomes,

$$\begin{cases} b_0 + b_1 \cdot g_i + \cdots + b_n \cdot {g_i}^n = k_i \\ b_0 + b_1 \cdot 1 + \cdots + b_n \cdot 1^n = w_1 \\ \vdots \qquad \vdots \qquad \vdots \\ b_0 + b_1 \cdot n + \cdots + b_n \cdot n^n = w_n \end{cases}$$

where $g_i = g(k_i, M)$, $w_i = w(i)$, for $i = 1, \ldots, n$. Similarly, the Equations 1 and 2 can be summarized as,

$$\begin{cases} a_0 + a_1 \cdot g_i + \cdots + a_n \cdot {g_i}^n = k_i \\ a_0 + a_1 \cdot 1 + \cdots + a_n \cdot 1^n = w_1 \\ \vdots \qquad \vdots \qquad \vdots \\ a_0 + a_1 \cdot n + \cdots + a_n \cdot n^n = w_n \end{cases}$$

where $a_0 = K$ and $g_i = g(k_i, M)$, $w_i = w(i)$, for $i = 1, \ldots, n$. From the above two equations, we have:

$$\begin{cases} (b_0 - a_0) + (b_1 - a_1) \cdot g_i + \cdots + (b_n - a_n) \cdot {g_i}^n = 0 \\ (b_0 - a_0) + (b_1 - a_1) \cdot 1 + \cdots + (b_n - a_n) \cdot 1^n = 0 \\ \vdots \qquad \vdots \qquad \vdots \\ (b_0 - a_0) + (b_1 - a_1) \cdot n + \cdots + (b_n - a_n) \cdot n^n = 0 \end{cases} \tag{9}$$

This results in:

$$\begin{vmatrix} 1 & g_i & \cdots & {g_i}^n \\ 1 & 1 & \cdots & 1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & n & \cdots & n^n \end{vmatrix} \neq 0 \quad \Longrightarrow \quad a_j = b_j,\ j = 0, 1, \ldots, n$$

In other words, $a_j = b_j$, $j = 0, 1, \ldots, n$, if and only if the determinant of Equation 9 is non-singular. Since we have $g_i > n$, $i = 1, \ldots, n$, and because $p$ (the modulo reduction) is prime, the above determinant is non-singular, and therefore, $a_j = b_j$, $j = 0, 1, \ldots, n$. This proves that $b_0 = K$ is the real key which was chosen by Alice. Hence, $k_i$ is an applicable password, and so, $(\Delta \cap \mathcal{K}) \subseteq \Gamma$. This implies that $m$ $(= |\Gamma|)$ is greater than or equal to the number of passwords chosen from $\mathcal{K}$ $(= |\Delta \cap \mathcal{K}|)$. □

## References

[Bakhtiari et al. 95a] Bakhtiari, S., Safavi-Naini, R., and Pieprzyk, J.: Cryptographic Hash Functions: A Survey; Technical Report 95-09, Department of Computer Science, University of Wollongong, 1995.

[Bakhtiari et al. 95b] Bakhtiari, S., Safavi-Naini, R., and Pieprzyk, J.: Keyed Hash Functions; *Cryptography: Policy and Algorithms Conference*, vol. 1029 of *Lecture Notes in Computer Science (LNCS)*, pp. 201–214, Springer-Verlag, July 1995.

[Berson et al. 93] Berson, T. A., Gong, L., and Lomas, T. M. A.: Secure, Keyed, and Collisionful Hash Functions; Technical Report (included in) SRI-CSL-94-08, SRI International Laboratory, Menlo Park, California, 1993; The revised version (September 2, 1994).

[Carter and Wegman 79] Carter, J. L. and Wegman, M. N.: Universal Class of Hash Functions; *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[Damgård 89] Damgård, I. B.: A Design Principle for Hash Functions; In *Advances in Cryptology, Proceedings of CRYPTO '89*, pages 416–427, 1989.

[Gong 95] Gong, L.: Collisionful Keyed Hash Functions with Selectable Collisions; *Information Processing Letters*, 55:167–170, 1995.

[M. Naor and M. Yung 89] M. Naor and M. Yung: Universal One-Way Hash Functions and Their Cryptographic Applications; In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 33–43, 1989.

[Preneel 93] Preneel, B.: *Analysis and Design of Cryptographic Hash Functions*; PhD thesis, Katholieke University Leuven, 1993.

[Rivest 92] Rivest, R. L.: The MD5 Message-Digest Algorithm; RFC 1321, Apr. 1992. Network Working Group, MIT Laboratory for Computer Science and RSA Data Security, Inc.

[Wegman and Carter 81] Wegman, M. N. and Carter, J. L.: New Hash Functions and Their Use in Authentication and Set Equality; *Journal of Computer and System Sciences*, 22:265–279, 1981.

[Zheng et al. 93] Zheng, Y., Hardjono, T., and Pieprzyk, J.: The Sibling Intractable Function Family (SIFF): Notion, Construction and Applications; *IEICE Trans. Fundamentals*, E76-A(1), 1993.