Security evaluation of the Gordian algorithm

Citation for published version (APA):

Document status and date:
Published: 01/01/2000

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Security evaluation of the Gordian algorithm

P.L.A. Roelse
L.A.M. Schoenmakers

November 2000
Security evaluation
of the Gordian
algorithm

Confidential

Status: Version 1.1
Date: November 23, 2000

Contract No. WDC 275/JM/mw
ABN AMRO
### Approval of document

<table>
<thead>
<tr>
<th></th>
<th>Signature:</th>
<th>Date:</th>
</tr>
</thead>
</table>
| TU/e, Coding & Cryptology Group  
Prof. dr. ir. H.C.A. van Tilborg | [Signature] | 21-11-00 |

### Authors

<table>
<thead>
<tr>
<th></th>
<th>Signature:</th>
<th>Date:</th>
</tr>
</thead>
</table>
| TU/e, Coding & Cryptology Group  
Dr. ir. P.L.A. Roelse | [Signature] | 17-11-00 |
| TU/e, Coding & Cryptology Group  
Dr. ir. L.A.M. Schoenmakers | [Signature] | 17/11/00 |
1 Introduction

This document describes a security evaluation of the Gordian algorithm, as defined in [1]. The algorithm is used in a user authentication mechanism, the so-called Gordian Access Key system. A brief overview of this system is given in Section 1.1. The purpose of this report is to identify strengths and potential weaknesses of the Gordian algorithm.

1.1 System overview

The Gordian Access Key (GAK) system is a user authentication system, providing a mechanism for the unique identification of computer system users to a host computer. The GAK system can be integrated in existing computer systems. An operational GAK system consists of host-resident software and a number of hand-held Access Key devices (one for each user), also referred to as tokens.

If a user wants to log-on (i.e. identify him/herself) to the host, the corresponding unique user ID is entered on a terminal, see also Figure 1. This user ID is sent to the host, which generates a challenge and sends this back to the terminal. The challenge, also referred to as the stimulus, contains a pseudo-random number which is used for one log-on session only. From the terminal, the user transfers this challenge to his/her token. The token then computes a response from this challenge and user unique (secret) information stored inside the token. The algorithm used for this computation is called the Gordian algorithm and is described in detail in [1]. The response, also
referred to as a one-time password or answer, is displayed on the screen of the token, after which the user can enter it on the terminal. It is then transferred to the host, which performs the same computations on the challenge as the token of the user. For this purpose, the Gordian algorithm is also available on the host, as well as a security database containing the user unique (secret) information corresponding to the user ID. Access to the host system is only gained if the computed response equals the received response, in which case the identification is successful.

1.2 Document overview

This document contains a security evaluation of the Gordian algorithm used for the computation of the response from the challenge and the user unique secret information (also referred to as the key). Recall from Section 1.1 that this algorithm and key is available in both the token of the user and in the host computer. This report does not contain an evaluation of the system, such as the (physical) protection of the host computer and token and the information stored in these. An evaluation of the GAK system can be found in [2].

This document is organized as follows:

Section 2 – Algorithm description. This section contains a description of the building blocks of the Gordian algorithm. The description is given in a mathematical notation that is needed for the evaluation, instead of the implementation oriented description in [1]. Although this document is intended to be self-contained, details of the algorithm that are not relevant for this security evaluation will be omitted.

Section 3 – Security evaluation. In this section the resistance offered by the Gordian algorithm against general attack methods are described. The attack methods include a dictionary attack, a random response attack and an exhaustive key search.

Section 4 – Conclusions. This section presents the conclusions of the security evaluation of the Gordian algorithm and its implications on the (practical) security of the Gordian Access Key System.

Section 5 – References. In this section pointers to the literature that was used during the project are listed.
2 Algorithm description

This section contains a brief description of the algorithm. The structure of the algorithm is given together with the details that are relevant for the security evaluation in Section 3. A complete description of the algorithm can be found in [1].

2.1 Notations and definitions

In the description of the Gordian algorithm and its evaluation, the following notation will be used. Let $Z^n_2$ be defined as the set of all binary vectors of length $n$ $(n \geq 1)$ with the addition $\oplus : Z^n_2 \times Z^n_2 \rightarrow Z^n_2$, also referred to as an exclusive-or (XOR). This addition is defined as a coordinate-wise addition modulo 2. For example, $(0,1,1)$ and $(1,0,1)$ are elements of $Z^3_2$ and $(0,1,1) \oplus (1,0,1) = (1,1,0)$. The elements (also called bits) of a vector $x \in Z^n_2$ are numbered from zero to $n-1$ from right to left, i.e. $x = (x_{n-1},x_{n-2},x_{n-3},...,x_0)$. The symbol $\ll$ is used to denote a concatenation of vectors, e.g. if $x = (x_3,x_2,x_1,x_0) \in Z^4_2$ and $y = (y_4,y_3,y_2,y_1,y_0) \in Z^5_2$ then $(x \ll y) = (x_3,x_2,x_1,x_0,y_4,y_3,y_2,y_1,y_0) \in Z^9_2$. Further, let $Z[x]$ denote the ring of polynomials with coefficients in $Z_2$. For example, $x^7 \oplus x^3 \oplus x \oplus 1$ is such a polynomial. If $g \in Z[x]$ with $g \neq 0$, then $Z[x]/(g)$ denotes the ring of residue classes of $Z[x]$ modulo $g$. It can be represented by the arithmetic of binary polynomials reduced modulo $g$.

2.2 The Gordian algorithm

The input for the Gordian algorithm is a 23 bit challenge $C$ (the last 23 bits of the Stimulus in [1]) and a token unique secret key $K$ of 120 bits. This key is the concatenation of the following four parameters:

- TOMORROW: this is a 32 bit vector which will be referred to as the Master Key MK in this document, so $MK \in Z^{32}_2$.
- ROLL_PARITY: this is a single bit (i.e. an element of $Z_2$) and will be referred to as $r$.
- ROOT: this is a 23 bit vector, referred to as the Whitening Key WK ($WK \in Z^{23}_2$) in this document.
- TODAY: this is a key of 64 bits, which is called the Session Key SK ($SK \in Z^{64}_2$) in the following. The reason for this name is that this key changes every couple of days under influence of MK, i.e. only remains valid for this period, which is viewed as a session.

With this notation $K := (MK \| r \| WK \| SK) \in Z^{120}_2$. The output of the algorithm is a 24 bit response $R$ (called the ANSWER in [1]). The block diagram of the algorithm is depicted in Figure 2 with $D \in Z^{23}_2$ a constant defined as $D := (0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)$.
The part within the dashed lines is executed every three days. It consists of updating the session key with the use of the Master Key and the function $R$ and the bit $r$ by using the function $T$. This bit is used for time synchronization of the token and the host between two consecutive 72 hour timeframes.

![Block diagram of the Gordian algorithm](image)

**Figure 2: Block diagram of the Gordian algorithm**

After the input is ‘whitened’ by the vector $WK \oplus D$, the function $F$ is applied, called the **major scrambling step** in [1]. This function is the heart of the algorithm and consists of repeating a round function a (large) number of times, together with a key scheduling algorithm for generating the round keys from the session key. Finally, the function $CP$ is applied to the output of $F$, which consists of appending the bit $r$ to the output and applying a coordinate permutation. In the remainder of this section, these functions will be described in a little more detail. Only aspects that are relevant for the security evaluation are given, a detailed description of the algorithm can be found in [1].

### 2.2.1 The function $F$

The function $F$ is a mapping from $Z_2^{64} \times Z_2^{23}$ to $Z_2^{23}$. Its first input argument is the Session Key $SK^{(i)}$, valid for the corresponding three day time window. Its second input argument is the output of the whitening step, i.e. $C \oplus WK \oplus D$. The output is denoted by

$$F(SK^{(i)}, C \oplus WK \oplus D).$$

The function is based on a Feedback Shift Register consisting of 23 memory cells of one bit each, the values of these cells are denoted by $f_0$, $f_1$, ..., $f_{22}$ respectively, using a non-linear feedback function. In this document only the feedback function is described, for a detailed description of the function $F$, the reader is referred to [1]. The basic step in the algorithm consists of clocking the
Feedback Shift Register once, i.e. first computing the output $b$ of the non-linear feedback function and then assigning $f_i \leftarrow f_i + b$ for $i = 0, 1, 2, \ldots, 21$ and $f_{22} \leftarrow b$.

The feedback function is depicted in Figure 3. All arrows in this picture represent single bits. The symbol $\otimes$ is used to denote a multiplication modulo two.

The value of Carry remains constant in the inner loop of the algorithm defining the function $F$, which consists of clocking the shift register 22 times (see [1]). After that, Carry is updated (outer loop, which is performed 225 times). All other input values of the feedback function depend on the round number. The key bits $k_0$, $k_1$, and $k_2$ are the round key bits derived from the Session Key $SK^{(i)}$ by a (simple) key scheduling algorithm. All other inputs are the contents of the corresponding memory cells of the shift register at that point in time.

For each of the four possible choices for the key bits $k_0$ and $k_1$, the Algebraic Normal Form (ANF) of the output bit $g$ as a function of the input bits is given by:

\[
\begin{align*}
(k_0, k_1) &= (0, 0): g = f_1 f_6 f_{14} f_{21} \oplus f_1 f_4 f_{21} \oplus f_1 f_6 f_{21} \oplus f_1 f_{14} \oplus f_1 f_{21} \oplus f_1 \oplus f_{21} \\
(k_0, k_1) &= (0, 1): g = f_1 f_{14} f_{22} \oplus f_1 f_4 f_{19} \oplus f_1 f_{14} f_{22} \oplus f_1 f_{19} f_{22} \oplus f_1 f_{22} \oplus f_1 f_{19} \oplus f_22 \oplus f_1 \oplus f_{22} \oplus f_1 \oplus f_{21} \\
(k_0, k_1) &= (1, 0): g = f_3 f_8 f_5 f_{12} f_{21} \oplus f_3 f_8 f_{12} f_{21} \oplus f_3 f_8 f_{12} \oplus f_3 f_{21} \oplus f_5 f_{21} \\
(k_0, k_1) &= (1, 1): g = f_3 f_8 f_{12} f_{19} f_{22} \oplus f_3 f_8 f_{12} f_{19} \oplus f_3 f_8 f_{12} f_{22} \oplus f_3 f_{19} f_{22} \oplus f_1 \oplus f_{22} \oplus f_1 \oplus f_{21}
\end{align*}
\]

These functions can be easily implemented in hard-/software, see also the software description in [1].
2.2.2 The function $R$

The function $R$ is a mapping from $\mathbb{Z}_2^{32} \times \mathbb{Z}_2^{64}$ to $\mathbb{Z}_2^{64}$, its first input argument is the Master Key MK, its second input argument the Session Key $SK^{(i-1)}$ ($i = 1, 2, ...$). The output is the Session Key $SK^{(i)}$, to be used for the next three day timeframe:

$$SK^{(i)} = R(MK, SK^{(i-1)}), \ i = 1, 2, ...$$

For the first 72 hours the Session Key is defined as $SK^{(0)} := SK$. The function $R$ is given in [1]. It is easily seen that this function can be described as follows. Let $SK^{(i)} := (s_{63}^{(i)}, s_{62}^{(i)}, s_{61}^{(i)}, ..., s_0^{(i)})$ for $j = 0, 1, 2, ...$ and $MK := (m_{31}, m_{30}, m_{29}, ..., m_0)$, then define the polynomials $s^{(i)}(x)$ and $m(x) \in \mathbb{Z}_2[x]$ as

$$s^{(i)}(x) := \sum_{i=63,62,..,0} s_i^{(i)} x^j, \text{ for } j = 0,1,2,...$$

$$m(x) := x^{64} \oplus \sum_{i = 63,62,..,32} m_{i-32} x^i \oplus \sum_{i = 31,30,..,1} (m_i \oplus 1) x^i \oplus 1.$$ 

The function $R$ then corresponds to a multiplication with the residue class of $x^{72}$ in the ring $\mathbb{Z}_2[x]/(m(x))$, i.e. $s^{(0)}(x) = s^{(i)}(x)x^{72} \mod m(x)$.

2.2.3 The functions $T$ and $CP$

The input for the function $T: \mathbb{Z}_2 \rightarrow \mathbb{Z}_2$ is the bit $r^{(i-1)}$, its output the bit $r^{(i)}$. The function is defined by $T(0) = 1$ and $T(1) = 0$, i.e. the output bit is the complement of the input bit (called a toggle in [1]). The function is called every 72 hours, the first 72 hours the bit $r$ of the key is used (i.e. $r^{(0)} := r$).

The function $CP: \mathbb{Z}_2^{23} \times \mathbb{Z}_2^{2} \rightarrow \mathbb{Z}_2^{24}$ has two input arguments, the output of the function $F$ and the bit $r^{(i)}$. The function concatenates these two vectors and then applies a coordinate permutation on the bits of that vector. This permutation is defined in [1]. It will not be given in this document, as it has no cryptographic meaning and is therefore irrelevant for the security evaluation. The output of $CP$ is the 24 bit response $R$. Note that

$$R = CP(F(SK^{(i)}), C \oplus WK \oplus D), r^{(i)}),$$

when written in the notation that was introduced above.
3 Security evaluation

3.1 Introduction

This subsection contains some general remarks concerning the cryptographic aspects of the Gordian algorithm. From the algorithm description given in Section 2, the following points are important to mention (see also Figure 2):

1. The number of bits of the input and output of the Gordian algorithm equal 23 and 24 respectively, which is small compared to the key size of 120 bits. Further, the input contains a considerable amount of redundancy; five bits synchronization pattern and six bits CRC check.
2. The constant D as used in the whitening step (called 'input scrambling' in [1]) has no cryptographic significance. Note that the bits of the Whitening Key WK are added modulo 2 to this constant, implying that the result can be replaced by a different Whitening Key WK' := WK ⊕ D. Knowledge and randomness of bits of WK' and WK are exactly the same.
3. The 120 bit key K is split into four different keys, i.e. K := (MK || r || WK || SK). The role of the four subkeys MK, r, WK and SK in the algorithm is rather 'independent'.
4. The function CP (called 'permutation/small scrambling' in [1]) has no cryptographic significance. The bit r(0) can be read off immediately from a response, after which its value for all other time windows is known. For any given response, the function CP can be undone by applying its inverse to the response, without further knowledge of key bits.

These properties of the Gordian algorithm will be exploited in the specific attack methods as described in the following subsections.

3.2 Dictionary attack

A dictionary attack (see e.g. [3]) requires 2^{12} (= 4096) challenge-response pairs under a fixed Session Key SK^{0}. This number is smaller than the block size of the Gordian algorithm suggests (2^{23} challenge-response pairs). The reduction is caused by the fact that only the 12 pseudo-random bits of the challenge are free to choose, the other bits of the challenge are either fixed (the synchronization pattern) or depend in a deterministic way on the 12 pseudo-random bits (the CRC check). This implies that knowledge of 4096 different challenge-response pairs for a fixed Session Key SK^{0} will allow to determine the response correctly to any possible challenge without knowledge of the key.

An adversary can also launch a dictionary attack with a dictionary that is not complete, i.e. contains less than 4096 different challenge-response pairs for a fixed key. In the following analysis, it is assumed that the challenges generated by the random number generator have a uniform distribution, i.e. each challenge is generated with a probability of 1/4096. Suppose that the dictionary contains k (1 ≤ k ≤ 4096) different challenge-response pairs. If the adversary makes m (
\( \geq 1 \) attempts to log-on, i.e. receives \( m \) challenges from the random number generator, then the probability \( p(m,k) \) that at least one of them is contained in the dictionary is given by

\[
(3.2.1) \quad p(m,k) = 1 - \left( \frac{(4096-k)}{4096} \right)^m.
\]

Note that this is the probability of successfully gaining access to the host by making \( m \) attempts to log-on and only giving a response if the challenge-response pair is in the dictionary, given that the dictionary contains exactly \( k \) different challenge response pairs.

**Example 3.2.1.** Table 1 contains approximations for the probability of success for some particular values of \( m \) (the number of attempts to log-on) and \( k \) (the size of the dictionary).

**Table 1: approximations for the probability \( p(m,k) \) for some values of \( m \) and \( k.**

<table>
<thead>
<tr>
<th>( m \backslash k )</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.0015</td>
<td>0.0029</td>
<td>0.0058</td>
<td>0.012</td>
<td>0.023</td>
<td>0.046</td>
<td>0.091</td>
<td>0.18</td>
<td>0.33</td>
<td>0.58</td>
</tr>
<tr>
<td>10</td>
<td>0.0049</td>
<td>0.0097</td>
<td>0.019</td>
<td>0.038</td>
<td>0.075</td>
<td>0.15</td>
<td>0.27</td>
<td>0.48</td>
<td>0.74</td>
<td>0.94</td>
</tr>
<tr>
<td>20</td>
<td>0.0097</td>
<td>0.019</td>
<td>0.038</td>
<td>0.075</td>
<td>0.15</td>
<td>0.27</td>
<td>0.47</td>
<td>0.72</td>
<td>0.93</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The pairs for the dictionary can be obtained by either a known or chosen plaintext (i.e. challenge) attack. A chosen plaintext attack requires access to the token of the corresponding user. Note that this will result in a dictionary valid for at most three days, by the change of the session key after that time. If a way can be found to eliminate the manual transfer of the 4096 responses of the token to the dictionary, a chosen plaintext attack will enable an easy construction of a complete dictionary. If an incomplete dictionary containing \( k \) challenge-response pairs is available, then Equation (3.2.1) can be used to determine the probability of gaining unauthorised access to the host. Note that this attack requires only temporary access to the token ('lunch-time' attack), after which access to the host can be gained for the rest of the three days the session key is valid. It is therefore important that the token is stored in a secure environment.

The known plaintext attack implies in practice the interception of communications between the token and the host, which is assumed to be a realistic scenario. Assume that an adversary has intercepted \( n \) (\( n \geq 1 \)) challenge-response pairs corresponding to one user and one three day time window. Note that this does not necessarily result in a dictionary containing \( n \) different challenge-response pairs, as intercepted pairs might be identical. Let \( r(n,k) \) denote the probability that the dictionary contains exactly \( k \) (\( 1 \leq k \leq \min\{4049,n\} \)) different challenge-response pairs, given that \( n \) challenge response pairs were intercepted. Further, let the number \( C(a,b) \) denote the usual binomial coefficient which is defined by \( C(a,b) := a!/(b!(a-b)!)) \). Then it can be shown that

\[
(3.2.2) \quad r(n,k) = \left( \frac{C(4096,k)}{4096^n} \right) \sum_{j=0,1, \ldots} (1)^j C(k,j) (k - j)^n / 4096^n.
\]

Now assume again that the adversary makes \( m \) attempts to log-on to the host. Let \( s(m,n) \) denote the probability that at least one of the \( m \) received challenges is in the dictionary, which can be used to gain unauthorised access to the host, given that \( n \) challenge-response pairs were intercepted. Using Equations 3.2.1 and 3.2.2 it follows that
(3.2.3) \( s(m,n) = \sum_{k=1,2, \ldots, n} \rho(m,k) r(n,k) \).

**Example 3.2.2.** Table 1 contains approximations for the probability of success for some particular values of \( m \) (the number of attempts to log-on) and \( n \) (the number of intercepted challenge-response pairs).

**Table 2: approximations for the probability \( s(m,n) \) for some values of \( m \) and \( n \).**

<table>
<thead>
<tr>
<th>( m ) ( n )</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.0015</td>
<td>0.0029</td>
<td>0.0058</td>
<td>0.012</td>
<td>0.023</td>
<td>0.046</td>
<td>0.089</td>
<td>0.17</td>
<td>0.31</td>
<td>0.53</td>
</tr>
<tr>
<td>10</td>
<td>0.0049</td>
<td>0.0097</td>
<td>0.019</td>
<td>0.038</td>
<td>0.075</td>
<td>0.14</td>
<td>0.27</td>
<td>0.46</td>
<td>0.71</td>
<td>0.92</td>
</tr>
<tr>
<td>20</td>
<td>0.0097</td>
<td>0.019</td>
<td>0.038</td>
<td>0.075</td>
<td>0.14</td>
<td>0.27</td>
<td>0.46</td>
<td>0.71</td>
<td>0.92</td>
<td>0.99</td>
</tr>
</tbody>
</table>

From Table 2 it follows that the number of log-on procedures for one user within each three day window should be small. In addition, an adversary should not be able to execute a large number of attempts to log-on to the host.

A successful dictionary attack allows access to the host without knowledge of the key. Due to the change of the session key every three days, the threat to the system is restricted to the corresponding three day timeframe, after which the attack has to be repeated. From the above it can be concluded that the Gordian algorithm does not offer an adequate protection against a dictionary attack. This is further weakened by the restriction on the number of different challenges. However, under the assumptions that

(i) an adversary is not able to execute a large number of attempts to log-on to the host,

(ii) tokens are stored securely and

(iii) the number of log-on procedures by each legitimate user in the corresponding three day timeframe is restricted,

a dictionary attack will pose no threat to the general system in practice.

**Remark 3.2.1 (Denial-of-service attack).** Putting restrictions on the number of invalid log-on procedures, as suggested by assumption (i), makes the system vulnerable to a ‘denial-of-service’ attack. I.e., if an adversary tries to log-on a number of times that exceeds the maximum amount of times allowed by the system without success, the service will also become unavailable to the corresponding legitimate user.

**Remark 3.2.2 (Aborting the log-on procedure).** It is important to treat an aborted log-on procedure as an invalid log-on. Otherwise the procedure could be aborted by an adversary whenever a challenge was received that was not in the dictionary.

**Remark 3.2.3 (Maximum number of attempts).** In the analysis above, it is assumed that an adversary can perform a maximum of \( m \) attempts in the corresponding three day time window. If the number of invalid log-on procedures is reset after each successful log-on by the legitimate user, the number of attempts which can be made by an adversary can be considerably larger (and consequently also his/her probability of gaining unauthorized access to the host).
3.3 Random response attack

Note that with the knowledge of an existing User ID, a log-on procedure for that user to the host can be started. Without any further knowledge, the probability of randomly guessing the correct response to the challenge offered by the host is about $2^{-24}$, implying that on average $2^{24}$ attempts will lead to a successful log-on and unauthorised access to the host.

The Gordian algorithm does not offer a sufficient level of protection against this kind of attack, implying that other countermeasures in the system should be taken to prevent this. Note the difference with a dictionary attack, as no correct challenge-response pairs of the user have to been known in this case. However, under assumption (iii) as stated in Section 3.2, a random response attack will be no problem in practice.

3.4 Exhaustive key search

The length of the key of the Gordian algorithm equals 120 bits. An exhaustive key search is a general attack requiring only a small number of known plain-ciphertext pairs (i.e. challenge-response pairs). The key size of the algorithm suggest that the workload of such an attack is around $2^{120}$, roughly consisting of one 'encryption' of a challenge under all possible keys and checking if the computed response equals the known one. Note that this attack results in finding the key of the corresponding user, from which all (session) keys to be used in the future can be computed. This means that, in contrast to the attacks described in the previous subsections, an exhaustive key search has to be performed only once for each user (or token).

As mentioned in Section 3.1, the 120 bit key $K$ is split into four different parts: $K = (MK || r || WK || SK)$ (see also Section 2.2). As already mentioned in Section 2.2.3, the bit $r^0$ can be read off immediately from a response, and can therefore not be considered to be secret. Further, knowledge of one $r^j$ enables the computation of all this bit for all time windows, in particular $r$. However, since this is only one bit, this is not considered to be serious.

Recall from Section 2.2.2 that two successive session keys satisfy the following relation (in terms of polynomials):

\[(3.4.1) \quad s^{(j+1)}(x) = s^0(x)x^{72} \mod m(x), \text{ for } j = 0,1,2, \ldots\]

This expression represents the function $R$ and is easily seen to be a one-to-one mapping for a fixed choice of $MK$, since $\gcd(m(x),x) = 1$, i.e. $x$ is a unit in the ring $\mathbb{Z}_2[x]/(m(x))$. In itself this is a good property, as no information of the session key is lost. On the other hand, Equation 3.4.1 implies that

\[s^{(j+1)}(x) \oplus s^0(x)x^{72} = m(x)l^0(x), \text{ for } j = 0,1,2, \ldots\]

for a certain $l^0(x) \in \mathbb{Z}_2[x]$ with $\deg(l^0(x)) < 72$. This implies that if a (small) number of pairs of successive session keys are known, the master key can be easily determined as $m(x)$ equals the greatest common divisor of the expressions $s^{(j+1)}(x) \oplus s^0(x)x^{72}$ for those values of $j$ that they are known with high probability. If not enough successive pairs are known, factoring the greatest
common divisor will in general leave only a couple of choices for $m(x)$. For one pair this corresponds to factoring the polynomial $s^{(i+1)}(x) \oplus s^{0}(x)x^{72}$. The redundancy in $m(x)$ (Section 2.2.2) can be used to identify the correct choice for $m(x)$.

Further, recall from Section 2.2.3 that a challenge and its response satisfy the relation $R = CP(F(SK^0, C \oplus WK \oplus D), r^0)$. From the fact that $CP$ and $F$ are invertible functions for any fixed choice of the key, it follows that two challenge-response pairs $(C_1, R_1)$, $(C_2, R_2)$ generated with the same key satisfy the following equation:

$$ (3.4.2) \quad C_1 \oplus C_2 = F^I(SK^0, CP^I(R_1)) \oplus F^I(SK^0, CP^I(R_2)). $$

Note that this relation does not depend on the Master Key MK and the Whitening Key WK. Under the assumption that $F^I$ behaves like a random mapping when a wrong (session) key is substituted in Equation 3.4.2, the relation holds with probability one for $SK^0$ and with probability $2^{-33}$ for all other choices for the session key.

These two observations suggest the following approach for an improved exhaustive key search (for one fixed User-ID):

1. Collect a small number of challenge-response pairs within the three day time window $j$, which can be recognized by inspection of the bit $r^0$, as it is contained in the responses and remains constant within the three day time window. Call these pairs $(C_1, R_1)$, $(C_2, R_2)$ etc.
2. Perform an exhaustive key search over all $2^{64}$ possible values for $SK^0$ to find the keys satisfying the relation $C_1 \oplus C_2 = F^I(SK^0, CP^I(R_1)) \oplus F^I(SK^0, CP^I(R_2))$. For each key that satisfies this relation, check if other relations like e.g. $C_2 \oplus C_3 = F^I(SK^0, CP^I(R_2)) \oplus F^I(SK^0, CP^I(R_3))$ hold. Under the assumption that the algorithm behaves like a random mapping if the wrong key is chosen, only a couple of known challenge-response pairs are sufficient to determine $SK^0$ uniquely.
3. Compute $WK = F^I(SK^0, CP^I(R_1)) \oplus C_1 \oplus D$.
4. Repeat Step 1 and 2 for the three day time window $j+1$ to determine $SK^{j+1}$.
5. Factor the polynomial $s^{(i+1)}(x) \oplus s^{0}(x)x^{72}$ in irreducible factors and try to construct $m(x)$ from these factors. If the number of possibilities to construct $m(x)$ is too large (which is unlikely in practice), repeat Step 1, 2 and 4 for different three day time windows $k, l, ...$ to collect a small number of equations $s^{(i+1)}(x) \oplus s^{0}(x)x^{72} = m(x)x^{72}$, $s^{(i+1)}(x) \oplus s^{0}(x)x^{72} = m(x)x^{72}$, ... and compute $\gcd(s^{(i+1)}(x) \oplus s^{0}(x)x^{72}, s^{(i+1)}(x) \oplus s^{0}(x)x^{72}, s^{(i+1)}(x) \oplus s^{0}(x)x^{72}, ...)$. Computer simulations in Mathematica show that in general two to three equations are sufficient to find $m(x)$ with high probability (and consequently MK). An alternative for repeating Step 1,2 and 4 is to perform an exhaustive search over all $2^{32}$ possible choices for $m(x)$. With high probability there will be a unique solution for $s^{(i+1)}(x) = s^{0}(x)x^{72} \mod m(x)$.

This attack can even be applied without knowing the challenges (at the cost of knowing slightly more responses), as the challenges contain redundancy (synchronization pattern, CRC).

To estimate the complexity of the algorithm described above, note that Step 2 requires a workload of approximately $2^{65}$ 'decryptions' (i.e. computing the response for a given key and challenge). The expected workload equals $2^{64}$ 'decryptions', as the correct key is expected to show up after searching half of the key space. Note that Step 2 is performed again in Step 4 of the algorithm.
This implies that Step 1 to 4 have an expected/worst case workload of $2^{65}$ and $2^{66}$ 'decryptions' respectively. If MK in Step 5 is found by either factoring the polynomial $s^{(i+1)}(x) \oplus s^{(i)}(x)x^2$ or by exhaustive search, then the complexity of this step is negligible compared to the complexity of Step 2 and the expected/worst case workload of the improved exhaustive key search algorithm therefore equals $2^{65}$ and $2^{66}$ 'decryptions' respectively.

Remark 3.4.1 (Implementation of an exhaustive key search). An exhaustive key search can always be divided in a number of tasks, where each task consists of searching the key in an independent area of the key space. Notice that no communications between the different tasks are needed. For software implementations, a well-known and powerful method is to divide the work using the internet. For dedicated hardware implementations, one can simply manufacor a (large) number of identical and simple hardware circuits. The key scheduling of the Gordian algorithm is simple compared to other computations performed in the algorithm, enabling a change of the (session) key at negligible cost in both software and hardware.

Example 3.4.1 (Software implementation). The Gordian algorithm was implemented in the C programming language and in JAVA (for reference). The C implementation was optimized for speed and computes challenges from responses at a rate of approximately $2^{10}$ per second using an Intel Pentium II 266 Mhz processor. Under the assumption that $n$ computers are available at the same time and that each of them searches at a rate of $2^9$ session keys per second in an independent area of the key space, the maximum amount of time $T_{sw}(n)$ needed for the exhaustive key search in software is given by $T_{sw}(n) = 2^{56}/n$ seconds. It can be easily verified that under these assumptions, the amount of time needed for realistic values for $n$ will be too large to be within practical range at this moment.

Example 3.4.2 (Dedicated hardware implementation). With dedicated hardware the exhaustive key search can be performed considerably faster. As an example, assume that a circuit is built that computes $F^1(SK_0, CP^1(R_1)) \oplus F^1(SK_0, CP^1(R_2))$ for a given key and two responses. Further assume that for this purpose two hardware implementations of the round function of $F^1$ are available on the circuit. Note that the key scheduling has to be implemented only once. The gate complexity of such a circuit is expected to be low compared to e.g. the complexity of a hardware implementation of the round function of DES. Under the assumptions that (i) $n$ of these identical circuits are available, (ii) each circuit is clocked at a rate of 2 Ghz, and (iii) checking one key takes approximately 5000 clock cycles (roughly one cycle for each update of the feedback shift register), the maximum amount of time $T_{hw}(n)$ needed for the exhaustive key search in dedicated hardware is given by $T_{hw}(n) = 2^{56}.2500.10^{-9}/n$ seconds $= 2^{46.4}/n$ seconds $= 2^{50}/n$ days. Although millions of circuits will be needed to make the exhaustive key search practical under these assumptions, it is considered to be within practical range. Moreover, the circuits are identical, have a low gate complexity and are therefore not expensive to build (in large numbers).
4 Conclusions

The security evaluation of the Gordian algorithm leads to the following conclusions. The algorithm does not offer an adequate protection against a 'dictionary attack' (Section 3.2) and a 'random response attack' (Section 3.3). The vulnerability to these attacks is due to the small number of input/output bits and the restrictions on the input bits. This implies that other countermeasures in the system must be taken to avoid these attacks, e.g.:

(i) tokens are always stored in a secure environment,
(ii) the number of log-on procedures for a legitimate user in a three day time window has to be 'small',
(iii) an adversary is not able to execute a 'large' number of attempts to log-on to the host.

Given an acceptable risk level for giving unauthorized access to the host, and given the maximum number in either (ii) or (iii), the maximum acceptable number for the other can be computed using Equation 3.2.3 (or looked-up in Table 2). It is important to mention that requirement (iii) makes the system vulnerable to a 'denial-of-service' attack. Further, it is important that an adversary can not increase his/her number of attempts for gaining unauthorised access by aborting the procedure (i.e. this should be treated like an invalid log-on) and by waiting until the legitimate user has performed a successful log-on (i.e. the number of invalid attempts allowed should preferably be reset to the maximum at the start of a new three day time window, and not after each successful log-on). The attacks mentioned above apply to one user, the dictionary attack is valid for the three day time window in which it is performed, i.e. a dictionary corresponds to one user and one three day time window.

The improved exhaustive key search algorithm (see Section 3.4) has a workload at most $2^{66}$ 'decryptions'. This is far less than the theoretical upper bound implied by the key size, i.e. $2^{120}$. If dedicated hardware is available (built), an exhaustive key search is considered to be within practical range. Note that this attack has to be performed successfully only once for a user, as it reveals all his/her (future) key material.

This leads to the following overall conclusion. During the evaluation of the Gordian algorithm several weaknesses have been identified. Under the assumption that no dedicated hardware is available for an exhaustive key search and that adequate countermeasures in the system prevent dictionary and random-response attacks, these weaknesses will be hard to exploit in practice for the purpose of gaining unauthorised access to the host.
5 References

