Cryptanalysis of the cellular authentication and voice encryption algorithm

William Millan, Praveen Gauravaram
Information Security Research Centre,
Queensland University of Technology,
GPO BOX 2434 (126 Margaret Street),
Brisbane, QLD, 4001, Australia

millan,praveen@isrc.qut.edu.au

Abstract: This paper presents two methods for cryptanalysis of the CAVE algorithm, a four or eight round cryptographic algorithm currently used in mobile telephony. Our attacks demonstrate that CAVE is insecure (with any number of rounds) as a hash function for authentication or data integrity applications.
Keywords: cryptographic hash function, CAVE algorithm, reconstruction attack, list attack.
Classification: Science and engineering for electronics.

References

[1] TIA, "TR45.3 Appendix A to IS-54," February, 1992. Visit http://www.tcs.hut.fi/ helger/crypto/link/practice/mobile.html for this document.
[2] C.Wingert, M.Naidu,"CDMA 1xRTT Security Overview," August, 2002. Visit http://www.telecom.co.nz/binarys/cdma_security_overview.pdf for this document.
[3] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone "Chapter 9: Hash Functions and Data Integrity," HandBook of Applied Cryptography, CRC Press, pp. 321-383, 1997.
[4] W. Millan, "Cryptanalysis of the Alleged CAVE algorithm," Proc. The 1st International Conf. Information Security and Cryptology, Seoul,Korea, pp. 107-119, December, 1998.
[5] P. Gauravaram, W. Millan,"Improved Attack on the Cellular Authentication and Voice Encryption Algorithm," Proc. International workshop on Cryptographic Algorithms and their Uses, Gold Coast, Australia, pp. 1-13, July, 2004.

1 Introduction

CAVE is a cryptographic hash function primitive which is used in ANSI-41 wireless networks like Analog Mobile Phone Standard (AMPS),Time Division Multiple Access (TDMA), Code Division Multiple Access (CDMA one, CDMA2000) for authentication, data protection, anonymity and key derivation [1, 2]. The CAVE algorithm is intended to authenticate a legitimate subscriber to the wireless network and protect the network and customers of mobile phones from the cloning fraud. As a 128-bit hash function, it should satisfy the following practical security requirements. These properties are well discussed in [3].

- *pre-image resistance*: For a given 128-bit hash value of CAVE, it requires 2^{128} hashing operations of CAVE to find one pre-image mapping to that value.
- 2^{nd} *pre-image resistance*: For a given input and 128-bit hash value of CAVE, it requires 2^{128} hashing operations of CAVE to find another pre-image mapping to the same result.
- *collision resistance*: For a given 128-bit hash value of CAVE, it requires 2^{64} hash computations of CAVE to find two different inputs mapping to that hash value.

We present two novel techniques that can be used to attack the CAVE algorithm[1]. These attacks show that CAVE does not satisfy any of the above mentioned requirements to achieve practical security. The first reconstruction attack on CAVE shows that the security offered by CAVE-4 (resp. CAVE-8) is less than 12 bits (resp. less than 13 bits for CAVE-8) as it computes a pre-image for a given hash value in around 2^{11} hashing operations of the algorithm (resp. around 2^{13} for CAVE-8). The attack must be repeated approximately 2^{80} times to provide a good probability to obtain *a single* valid pre-image that has redundancy consistent with the input processing stage of the specific CAVE application. Hence the complexity of this attack on CAVE-4 is 2^{91} (resp. 2^{93} on CAVE-8). Already this is sufficient to consider CAVE broken. We then improve on this result with our second attack on CAVE, which is able to compute *all* of the possible 2^{48} pre-images with a complexity no more than 2^{72} hashing operations of CAVE-4 (resp. CAVE-8) algorithm.

With regard to the potential for attacking the real system, we may see that the Voice Privacy Mask (VPM) which is generated by several successive runs of 4-round CAVE, has 520 bits. This is greater than the 176 bits entropy maximum for any output of the CAVE algorithm, so there must be a considerable redundancy in the VPM. Knowing the VPM (which is easily obtained by frequency analysis of intercepted encrypted speech) should therefore (at least in principal) provide sufficient information to fix *uniquely* the CAVE input that generated it. Thus it should be possible, somehow, to break the whole system, recover the shared secret data SSD_A and SSD_B, and so on to recover the A-key (master key).

As an other case, authenticating a legitimate subscriber is the main application of CAVE. If different input values that hash to a given digest are found, it is possible to illegally program Electronic Serial Number(ESN) and Mobile Identification Number(MIN) into the mobile phone thereby providing a fraudulent customer with an access to the wireless network. When the

[1]The preliminary results of the two attacks were published in [4] and [5]

authentication fails, subscriber calls to the network would not be protected even by voice encryption.

This paper is structured as follows: In Section 2 we review the structure and operation of the CAVE algorithm. In Section 3 we present two attacks on the CAVE algorithm and finally we conclude the paper in Section 4 with some remarks.

2 The CAVE algorithm

A description of the CAVE algorithm can be obtained from [1]. For completeness, we provide a description of the algorithm here, introducing some notation and drawing attention to various aspects that affect security.

The main components of the algorithm are sixteen 8-bit data *registers* (sreg[0,1,...,15]), two 8-bit offsets *offset_1* and *offset_2* and a 32-bit Linear Feedback Shift Register (LFSR) with bytes labeled as $LFSR_A$, $LFSR_B$, $LFSR_C$ and $LFSR_D$. CAVE operates in four or eight rounds as per the requirements of a specific application with each round having 16 register update *phases*. For a round r, the register value i at the start of the round is $sreg[r][i]$ and the register value at the end of the round is $ereg[r][i]$. The round number in CAVE starts as one less than the desired number of rounds, and it is decremented to 0. $r = R-1, R-2, \ldots, 0$. The LFSR defines a primitive feedback polynomial whose feedback function is defined as:

$$L_{t+32} = L_t \oplus L_{t+1} \oplus L_{t+2} \oplus L_{t+22} \tag{1}$$

Application specific input processing determines how various mobile phone data is mapped into the initial contents. In every case, the LFSR is not allowed to be initialised as all zero. For each phase, CAVE uses bytes from the LFSR, the offsets and two 8*4 LUTs or SBoxes (each table has 256 nibble values where a “nibble” is a 4-bit value) to modify one of the registers. The offsets *offset_1* and *offset_2* act as pointers into the low and high CAVE tables which are represented as $CT\_low[\cdot]$ and $CT\_high[\cdot]$.

The steps that take place in the low segment of CAVE are expressed as:

$$\mathit{offset\_1} = \mathit{offset\_1}_{prev} + (LFSR_A \oplus sreg[i])) \bmod 256 \tag{2}$$

$$temp\_low = CT\_low[\mathit{offset\_1}] \tag{3}$$

and the steps for high segment are similar, with the exception that a separate (high-phase) offset byte, and the second LFSR byte ($LFSR_B$) are used. The segment operation in the CAVE algorithm is shown in Fig **1**.

The byte $\mathit{offset\_1}_{prev}$ represents the previous value of the offset byte (which in all applications is initialized as a fixed publicly known constant).

After the completion of a phase, the LFSR cycles once resulting in a minimum of sixteen LFSR shifts in each round of CAVE. Between rounds, bits in the registers are shuffled by using the low CAVE table to define a byte permutation followed by a 1-bit rotation on the 128-bit register block as a whole.

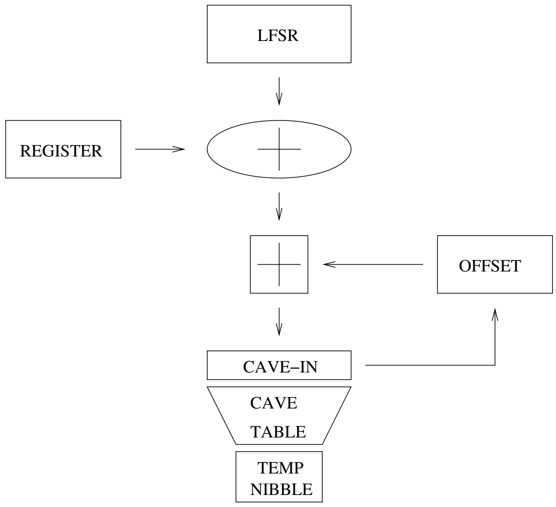

Fig. 1. Segment operation in CAVE

The data registers at the end of a round are represented as $ereg[0, 1, \ldots, 15]$. The end round bit permutation is represented as $sreg[r-1][0, 1, \ldots, 15] = BP[ereg[r][0, 1, \ldots, 15]]$, recalling that the round index is being decremented, so that round $r-1$ follows r.

A Sketch of the CAVE Algorithm

1. Establish initial data values by input processing.

2. For $r = 3$ down to 0 do:
   (a) For $i = 0$ to 15 do:
   i. Do Low Segment: get value of $temp_{low}$.
   ii. Do High Segment: get value of $temp_{high}$.
   iii. Set $temp = temp\_high \| temp\_low$.
   iv. Set $ereg[r][i] = sreg[r][i + 1(mod16)] \oplus temp$.
   v. Cycle LFSR once.
   vi. Next i.
   (b) Do Bit Permutation:$sreg[r-1][0, \ldots, 15] = BP[ereg[r][0, \ldots, 15]]$
   (c) Next r.

3. Output processing.

3 Two attacks on CAVE

In this Section we describe two different attacks on the CAVE algorithm. The reconstruction attack on CAVE shows that the security offered by CAVE-4(resp. CAVE-8) is less than 12 bits (resp. less than 13 bits for CAVE-8) as it computes a pre-image for a given hash value in around 2^{11} hashing operations of the algorithm (resp. around 2^{13} for CAVE-8). The list attack on CAVE can be used to compute all valid 2^{48} pre-images with a complexity no more than 2^{72} hashing operations of CAVE-4 (resp. CAVE-8) algorithm. As the number of rounds of CAVE is increased, the relative advantage of the list attack over the reconstruction attack also increases. The significant reduction in effort can make the second attack more threatening for CAVE in practice.

3.1 Reconstruction attack

The first step is to guess a 32-bit LFSR value and then run the known primitive feedback polynomial for sufficiently many cycles to specify all LFSR bytes at every stage of the CAVE calculation. Generating 200 cycles is enough. Given the LFSR values, the attack is attempting to discover values for the two offset bytes at the start of the last round which allow "self-consistent" re-construction of the input data registers.

Begin by guessing the values of 24 bits: both offset bytes and the value of $sreg[0]$ at the start of the last round. Given such a guess, the CAVE algorithm is then worked forward normally for one phase to determine the temp byte for phase $i = 0$, $temp[0]$. Now we have assumed the values of $ereg[0, \ldots, 15]$ are all known after reversing the last bit permutation, so we may calculate $sreg[1] = ereg[0] \oplus temp[0]$. This may be iterated for all phases in the last round, until a value for $temp[15]$ is obtained. Then we can calculate $ereg[15] \oplus temp[15]$ and compare it with the guess we made for $sreg[0]$ back at the start of the last round reconstruction. This is the "sanity check". If the values are equal, then our initial guess has led to a self-consistent reconstruction, and the attack may then pass to the last round, and so on. However, should the sanity check fail, we simply increment a counter of failed guesses for that round, and then guess again.

The last round of CAVE is treated differently from other rounds, in that it is reconstructed forward, rather than backwards. This is possible for the last round only, since the final offset values are not already fixed. When the attack proceeds to the second last (and previous) rounds, the final values of the offsets have been fixed, so the attack must work backwards from those values. We now examine the process of reverse reconstructing a segment in more detail. At the start of the backwards reconstruction of a round $r > 0$, we guess a value for $sreg[15]$ and propagate the effect of this choice backwards to obtain self-consistent values for $sreg[14, 13, \ldots, 0]$. Once $sreg[0]$ is available, we may compare it to $ereg[15] \oplus temp[15]$. As before a match means the reconstruction is successful.

Complexity analysis: Using this attack procedure, we found in experiments that 4-round CAVE can be broken in average time equivalent to $1.3 * 2^{10}$ executions of 4-round CAVE and 8-round CAVE can be broken in $1.25 * 2^{12}$ executions of the algorithm. Increasing number of rounds from 4 to 8 has only increased the workload by 8 times. This suggests that each extra round of CAVE adds less than 1 bit security, and hence increasing the number of rounds of CAVE is not an effective way to increase security. The theoretical upper bound for this attack is 2^{128} executions of CAVE. Our analysis shows that it takes around 2^{13} iterations to break 8-round CAVE.

3.2 The list attack on CAVE

In this section we present a second approach to attack CAVE. We first use a precomputation to establish look-up-tables that define the operation of a segment in CAVE. Then, given a 128-bit hash output (the final values of the register bytes), these tables are used to guide a process which maintain *lists of all* data that are self-consistent with respect to the sanity checks used in the previous attack. These lists are generated for consecutive segments within a phase, then consecutive phases within a round. Our experiments revealed that the resulting data sets can specify about half of the unknown LFSR bits! Similarly, the process may be extended across more phases back to the start of the algorithm. Considering the big picture, the CAVE algorithm hashes the fixed input of 176 bits down to 128 bits. So it is expected that each output to have 2^{48} preimages. We make a 24-bit initial guess each time, so we expect to have 2^{24} elements in each final list. To decrease practical running times, we first compute LUTs representing the set of the most frequently repeated operations in CAVE. These two LUTs consist of 24-bit entries, an output offset byte, a *temp* nibble and an extra cycle counter.

The overall attack algorithm can be described as follows.
Pre-computation: Calculate the high and low LUTs.

- **Init**: Repeat, for all 2^{24} values of $sreg[15]$ and the pair of offset bytes:
- **Step 1**: Use the LUTs to find lists of valid inputs to both segments in two consecutive phases.
- **Step 2**: For each phase, combine the two segment data lists into a list of valid data for that phase.
- **Step 3**: Combine the adjacent phase lists into a single list for the pair of phases.

Final: Combine the remaining lists, filtering for consistency, to determine the list of all possible valid inputs.

Complexity analysis: We may develop an upper bound for the complexity of our attack using the phase-equivalent complexity as the fundamental unit. Step 1 has complexity less than 2^{24} of these units, for each of the 2 phases in each of 2 adjacent segments making a total effort of 2^{26}. Step 2 requires around 2^{25} effort for each of the 2 phases, so that it makes 2^{26} effort as well, for a running total of 2^{27} phase-equivalent units. For the first time only, Step 3 must consider all pairings from two segment lists each of size 2^{16} elements, for a total complexity of 2^{32} operations. This dominates the complexity from the first two steps, so we may safely upper bound the complexity of finding all data consistent across two consecutive phases as being clearly less than 2^{33} phase-units. Lists become size of 2^{24}, so combining them costs 2^{48} effort. We use this as an upper bound on complexity for each phase in the second attack (64 phases in CAVE-4). As all these calculations must be performed 2^{24} times (with different initial choices for the pair of offset bytes and the start registers in the Init stage), so we expect the effort to find all valid data for 4-Round CAVE to be less than $2^{48} * 2^6 * 2^{24} = 2^{78}$ phase-units (which is about 2^{72} calculations of 4-Round CAVE which has 64 phases).

3.3 Comparison in complexities

The list attack on 4-Round CAVE compares favourably with the 2^{91} effort required by the first attack. The effort to extend this attack to 8-Round CAVE is minor: only another $2^{48} * 2^6$ effort for each of the 2^{24} trials is an extra 2^{78} phase-units or double the effort above what was needed to break 4-Round CAVE. To compare, the reconstruction attack requires eight times the effort.

4 Conclusion

In this paper we discussed two powerful attacks on the CAVE algorithm, casting serious doubt over the long term security of all CAVE applications. The second attack discussed in the paper may even threaten the security of real CAVE implementations. The decision to replace CAVE with Authenticated Key Agreement (AKA) was made in 1999. The slow standardization process, added to that slower adoption by the operators is delaying its replacement. Considering the threats we strongly recommend that where CAVE is still in use, it should be replaced with AKA as soon as possible.

Acknowledgments

The authors thank anonymous reviewers for their comments and suggestions which has helped in improving the content in [4, 5].