Attacking phone privacy

Karsten Nohl <karsten@srlabs.de>
Security Research Labs, Berlin

Abstract. GSM's encryption function for call and SMS privacy, A5/1, can be broken in seconds with 2TB of fast storage and two graphics cards. The attack combines several time-memory trade-off techniques and exploits the relatively small effective key size of 61 bits.

GSM Basics

Scope. GSM telephony is the world's most popular communication technology spanning most countries and connecting over four billion devices. Besides telephony, GSM is used for text messaging, Internet browsing and device-to-device communication. The standards for voice and text messaging date back to 1990 and have never been overhauled. In particular, vulnerabilities in GSM's original security design have never been patched. Data capabilities other than SMS messages were later added and use different security measures, which are not covered in this briefing.

Encryption. Phone calls and text messages can be encrypted between a phone and a base station. The base station decides whether encryption is used. Operators in the developed world typically use the original A5/1 encryption function, while a number of operators in the developing world do not use any encryption. The intentionally weak A5/2 cipher is being phased out globally while the stronger A5/3 cipher is not yet being used even though most new phones support it.

Key derivation. A phone's SIM card and the operator share a secret master key for encryption. A session key is derived from the master key and a random number for each call or text message[1]. Only this session key is provided to the base station by the operator's central system, so to not disclose the master key to roaming partners. An operator can use any hash function for the key derivation as long as the function is known to the SIM and the operator's central system. After the early standard for key derivation, COMP128v1 was shown to be invertible, most operators do not anymore disclose their choice of hash function, potentially creating security-through-obscurity. This briefing deals with the cipher steps after session key derivation, which are standardized. The method explained herein discloses an A5/1 session keys that can be used to decrypt one phone call or one text message.

Known plaintext. In order to verify a cryptographic key after cracking it, some of the encrypted data must be known. In our setup, we require chunks of consecutive 64 bits. These known bits are gathered from encrypted control messages. Commercial A5/1 crackers leverage the *Cipher Mode Complete* message, which is the first encrypted message in an encrypted transaction and usually

[1] Sometimes session keys are used for several calls and messages.

contains the same data, mostly empty padding bytes. Empty dummy frames are also encrypted even though they carry zero information, making them a prime target for key crackers. A recent GSM standard amendment requires this message and several others to be randomized, thereby taking away their potential for key cracking. However, other messages including the *System Information* messages also carry highly-predictable data that can be used for known plaintext where the *Cipher Mode Complete* message is not available.

Crypto Basics

Stream ciphers. A5/1 is a stream cipher that consumes a 64 bit secret key and maintains an equally large secret state derived from the key. Key bits are extracted from this secret state one bit at a time creating a stream of secret bits, called the *cipher stream*. Both the phone and the base station generate the same cipher stream to encrypt and decrypt messages. The sending party XORs the data to be sent with a section of the cipher stream and the receiving party XORs the same cipher stream to cancel out the obfuscation thereby decrypting the original data. As long as the A5/1 state stays secret, the data is exchanged confidentially. While algebraic attacks on the A5/1 cipher exists [1,2], none of them apply to the way A5/1 is used in GSM since the attacks require large segments of known plain text.

Brute force attacks. A5/1 can currently only be broken with generic attacks that exploit its relative small key size. The most straight-forward attack is a brute-force attack that tries all possible states to find the one that correctly decodes a packet. Brute-force is also the most expensive attacks, requiring—for every call to be decoded—thousands of CPU years or dozens of GPU[2] years even after applying the optimizations described below.

Code book attacks. A second generic attack computes the mapping between secret states and cipher streams for every possible state. The resulting code book is sorted by the cipher streams. Each secret state can be retrieved with a single look-up from the code book. While the initial cost of computing the code book are as high as a single brute force run, the incremental computational costs per crack are negligible. However, storing the code book of a 64 bit cipher is extremely costly requiring dozens of Petabytes.

Time-Memory Trade-Offs

Brute force (no storage, high attack time) and code book attacks (large storage, instantaneous) are two cornerstones of a design space of generic cryptographic attacks. In this design space, time can be traded for storage (ie, memory), both can be traded for higher attack success, and to some degree traded for shorter pre-computation time. The attack options between the cornerstones are called *time-memory trade-off* (TMTO) attacks.

Table look-up. TMTO attacks build upon the code book idea but use more complex data structures to store the mapping of secret states to cipher streams. In the simplest case, a constant number of state->output computations are chained together. Only the first and the last element of this chain are stored in a look-up table. Assume a table that stores the inputs and outputs of

[2] GPUs are highly parallelized processors on state-of-the-art graphics cards.

```
1:  $state = $input                    // typically chosen at random
2:  for n=1…1,000,000
3:        $output  = A5/1( $state)
4:        $state     = $output
```

A large table with many input-output pairs covers a significant portion of the state space, just like a code book. However, entries in the chains cannot directly be accessed since only the first and last element are stored. To find an $output_to_be_cracked in the table, the following simple algorithm is used that builds a chain starting from the look-up value to find the correct end value, and then computes the head of the chain starting from the matching $input value:

```
5:  $state = $output_to_be_cracked
6:  for n=1…1,000,000
7:         $end_value = look-up_ chain_end_values_from_table($state)   // compute chain …
8:         if( $end_value  != null )                                    // … until end point is found
9:                 $start_value = get_start_value ($end_value)          // Start chain from beginning
10:                $state2 = $start_value
11:                while( A51($state2) != $output_to_be_cracked)        // [ignores collisions]
12:                        $state2 = A51($state2)
13:                return $state2              // secret state that generates $output_to_be_cracked
14:        $state = A5/1 ($state)
```

This approach provides a continuous trade-off curve between time and storage. A look-up takes more time as the chains get longer (in this example all chains are of length 1,000,000), but also uses less storage. However, this naïve pre-computation algorithm is impractical for two reasons:

1. **Hard disk bottleneck:** 1,000,000 A5/1 computations are feasible in short time, but 1,000,000 storage look-ups are not.

2. **Collisions:** As the tables grow, values start reappearing at a high rate since the A5/1 mapping is not a perfect ring, leading to redundancy between two chains from the point of the collision until the chain ends. These collisions waste storage and prevent a single table from reaching sufficient coverage.

A number of techniques exist to overcome the practical limitations. The two most popular time-memory trade-off techniques are *distinguished points* [3] and *rainbow tables* [4]. A continuous trade-off also exists between these two techniques [5].

Distinguished points. This optimization solves the hard disk bottleneck. Only points that match a certain criterion – for example, having a certain number of zeros at the end – are stored as end values. The pseudo code above could be updated as follows:

```
2:  while( ( $state && 0xFFFF) != 0)  // test for 16 zero bits
```

```
7:  if(( $state && 0xFFFF) != 0) {
         $end_value = look-up_ chain_end_values_from_table($state)
         …
```

The chains will be of variable length but always end in a *distinguished point*. When cracking a value, a chain is computed until a distinguished point is reached and only that point in looked-up from the hard disk, thereby mitigating the hard disk bottleneck.

Distinguished points also help with finding collisions since redundant values lead to the same end point and can be filtered out.

Rainbow tables. A more effective measure against collisions are rainbow tables. This technique uses a different function (or *color*) for every chain link, ie:

```
3:  $output       = A5/1( $state ^ n)
```

Colliding values in these *rainbow tables* lead to only single values redundancy since the next links after the colliding values are computed with different colors (unless the collision occurs in the same link). Writing the pseudo look-up code for rainbow tables is left as an exercise for the curious reader.

Combined approach. The optimal table design for cracking A5/1 is a combination of distinguished points and rainbow tables. Chains are computed until a distinguished point is reached, then the rainbow function/color is changed, the computation continues with this second function until the next distinguished point is reached, the color is changed again – for a total of 8 colors in an optimized table layout.

To reach the necessary coverage, we compute several tables that work independently from one another. Each table uses a different set of colors which minimizes redundancy across the tables. The tables can be maintained on different machines providing a simple way for distributing the pre-computation and cracking costs.

Critical path length. A last optimization criterion in the table design is a trade-off between overall cracking cost and minimum crack time. Longer chains are more efficient within certain bounds. However, it takes longer to finish a longer chain on a single computing core, which lower-bounds the attack time.

All trade-off calculations are embedded in an Excel model found on the project web site[3].

Optimization

After tweaking the table parameters, our cracker was further optimized in three dimensions resulting in several orders of magnitude speed-up. Our implementation target are fast graphics card's GPUs, which offer better computing power for simple instructions per dollar and per watt than CPUs and are easier to source and program than FPGAs. Modern GPUs by NVidia and ATI offer several dozen computing cores providing a high level of parallelization out of the box.

GPU implementation: Bitslicing. Further parallelization is achieved through stuffing multiple data streams in one data word and computing an instruction on the entire data word (single instruction, multiple data: SIMD). In the extreme case, we take one bit of 256 ongoing computation streams and

[3] http://reflextor.com/Rainbow.Table.Parameters.xls

stuff those in a single 256 bit data word (which GPUs provide). The increased parallelization accelerates both the table pre-computation and the cracking of keys.

Cryptographic: Key space shrinking. The cost of cracking a cryptographic cipher using generic attacks scales with the secret state size (or key space) of that cipher. In case of TMTO tables, fewer or shorter chains are needed when the key space is smaller. For A5/1, we can shrink the key space from the original 64 bit by repeatedly clocking the cipher. Due to imperfect non-linear behavior, several states collide on the same state in each clock, thereby decreasing the number of existing states for each added round. In GSM, the A5/1 cipher is clocked 100 times before first used. Therefore, adding 100 clocks to each A5/1 computation during table computation and look-up does not lessen the applicability of the tables to GSM. These 100 clocks shrink the key state to 15% its original value, effectively taking almost 3 bits of key size away. As can be checked with the Excel model, the speed gain of this effect is around 30x.

Storage: Table compression. Several optimizations went into storing the start and end values as efficiently as possible. First, the start values are not computed randomly but rather generated by a simple pseudo-random functions. The inputs to this function are stored as start values, which are 34 bits in our case instead of 64 bit needed for a random state. A second optimizations leverages the fact that tables are sorted by end values. Instead of providing the entire value, the delta to the preceding value is stored. Only the first values in each block are stored in full. Since neighboring values only differ in the last bits, a complete chain can be encoded in 52 to 72 bits using variable length codes, with an average of around 54 bits per chain, effectively compressing the tables to 42% their original 2x64 bit format. An index is added for fast look-up from the complex table structure. The index is delta encoded to memory and has a footprint of 42MB per 47GB table, allowing the full tables set to be indexed from 2GB of RAM.

An open TMTO framework that implements these optimizations can be downloaded from the project web site[4].

Results

Combining all the optimizations, we are able to break A5/1 on commodity hardware.

Table generation. Our current table set contains 40 tables for a total of 2TB. Computing the set took about one month on four state-of-the-art graphics cards. About 850 kWh of power went into generating the tables.

Key cracking. Given two encrypted known plaintext messages (ie. *Cipher mode complete* and a *System Information* message), the table set finds a secret key with almost 90% probability. The cracking takes about five seconds on two GPUs. Roughly 100,000 look-ups are required which fast SSD disks can provide within five seconds. A large array of small USB sticks can potentially provide comparable speeds at lower costs.

[4] http://reflextor.com/trac/a51

Defenses

Cryptographic attacks in general and TMTO attacks in particular can be made more difficult through several standard techniques.

Avoid known plaintext. Transmitting known data appears useless in any event as the other side already knows what you are going to send. When encryption is used, known plaintext packets are even harmful. A well designed system only transmits unknown data. In cases where static packets cannot be prevented, these should at least be randomized. Randomization of the payload padding has been standardized for GSM. This quick fix that would cause deployed A5/1 crackers to stop working. However, the randomization is not widely used even though the required changes to base stations are small.

Block ciphers. While not strictly superior to stream ciphers, block ciphers have held up much better to cryptanalysis. All standard ciphers today including 3DES, AES, RSA and ECC are block ciphers. TMTO attacks are made increasingly difficult through the use of block ciphers since only blocks of ciphered data can be attacked as opposed to arbitrary segments of a continuous key stream. TMTO tables for a 64 bit block cipher need at least 64 times more plaintext packets to be as efficient as our TMTO tables for stream ciphers. For GSM, a 128 bit block cipher has been standardized as A5/3. No practical attacks are known against A5/3, which is also used in 3G/UMTS. However, roll-out costs of A5/3 for GSM are significant and no major operator has upgraded yet.

Conclusion

Security measures have a limited lifetime and the A5/1 cipher in GSM has exceeded its lifetime. Designed as a medium-security cipher during the cold war, is it no surprise that today's increasingly powerful computers can crack the 64 bit key in practical time. Combining cryptographic optimizations with techniques for highly parallel programming and smart storage allocation, A5/1 can even be cracked in under a minute with a single PC.

Short term protocol patches already exists that make cracking much harder by not disclosing known plaintext unnecessarily. These patched should be deployed with high priority. In the long term, GSM (2G) will not provide sufficient security and stronger alternatives such as UMTS (3G) and LTE (4G) should be preferred.

References

[1] *Real Time Cryptanalysis of A5/1 on a PC*. Alex Biryukov, Adi Shamir and David Wagner. FSE 2000.

[2] *Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication*. Elad Barkan, Eli Biham and Nathan Keller. CRYPTO 2003.

[3] *Cryptography and Data Security*, Dorothy E. Denning, Book 1982.

[4] *Making a Faster Cryptanalytic Time-Memory Trade-Off*. Philippe Oechslin. CRYPTO 2003.

[5] *Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers*. Alex Biryukov and Adi Shamir. CRYPTO 2000.