An Introduction to New Stream Cipher Designs

Tor E. Bjørstad

University of Bergen, Norway
Email : tor.bjorstad@ii.uib.no

1 What is a stream cipher?

Even with "nothing to hide", it is very often desirable to protect the privacy of our data and our communications. The usual way to do this is by using encryption[1]. It is safe to say that the Internet as we know it would not exist without strong crypto – whether it is used to protect remote logins, e-commerce transactions, the hard disk of your laptop, or something completely different. The goal is mostly the same in all cases: keeping the data or communications confidential.

Briefly speaking, a symmetric encryption algorithm takes some *plaintext* data and a secret *key* as input, and outputs a *ciphertext*. The goal of the exercise, of course, is that anyone who does not know the key, should not be able to deduce anything useful about the plaintext or the key from the ciphertext, while anyone who does know the key is able to decrypt and recover the plaintext from the ciphertext. There are two main classes of symmetric encryption algorithms, block ciphers and stream ciphers.

Block ciphers are the most well known type of symmetric encryption. Some famous algorithms are the old Data Encryption Standard (DES) and its replacement, the Advanced Encryption Standard (AES). These ciphers operate, as the name indicates, on fixed-length data blocks. AES takes 128 bits of plaintext as input together with a 128-bit secret key, and outputs 128 bits of ciphertext. In order to encrypt larger amounts of data securely, a chaining mode such as CBC must be used. Data fragments shorter than the block length usually have to be padded to the block length, increasing transmission overhead.

Stream ciphers are different. A stream cipher may, very loosely speaking, be thought of as a cryptographically secure pseudo-random number generator with some extra bells and whistles. These algorithms take the secret key as well as a public initialisation vector (IV, sometimes called a *nonce*) as input[2], and output a stream of random-looking symbols, known as the *keystream*. The

[1] It should be emphasised at this point that encryption alone is almost *never* enough by itself to build a secure protocol or application. Generally, one should *always* use a digital signature or a message authentication code (MAC) to maintain the integrity of encrypted data. Surprisingly often, it will be possible to attack an encrypted protocol without integrity checking, by tampering with the data in some particular way. However, this is a completely different cup of tea and far outside the scope of this short paper.

[2] The use of an IV makes it possible to encrypt several data streams without changing long-term secret keys. However, a given key/IV pair must only be used once.

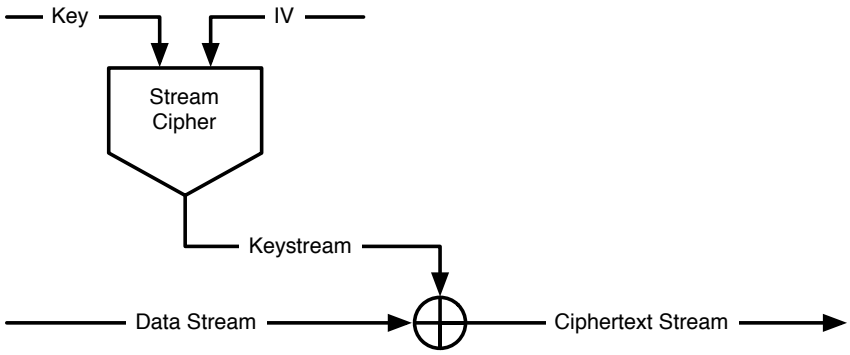

Fig. 1. Schematic representation of encryption with a stream cipher.

keystream symbols are usually either single bits, bytes, or machine words. To encrypt a data stream, one simply has to exclusive-or (XOR) the data symbols with the keystream. Decryption is of course the exact opposite, since the XOR operation is symmetric; when the ciphertext c is computed from the message m and keystream x as $c = m \bigoplus x$, we also have that $m = c \bigoplus x$. Real-world examples of stream ciphers include RC4 (used for WEP/WPA, by Bittorrent, and by SSL, to name a few) A5/1 (GSM telephony) and E0 (Bluetooth), as well as AES in some modes of operation (notably counter, or CTR-mode).

How does a stream cipher work? Although the specific details of stream ciphers vary immensely fro cipher to cipher, there is a certain amount of common structure. A stream cipher consists of a certain amount of internal state, which should be at least twice the size of the secret key to prevent certain generic attacks. Given a key and IV, the algorithm proceeds by a specified number of initialization steps, in which the key, IV and initial contents of the state are mixed in a nonlinear fashion. After this, the cipher starts outputting keystream symbols as a function of the (now sufficiently randomised) state, while continuing to mix and evolve the contents of the state itself. A modern stream cipher specification should make clear certain usage limits: how many keystream bits can be generated by a single key/IV pair, and how many IVs can be used before the secret key itself must be changed.

What are the advantages of stream ciphers over block ciphers? Stream ciphers may be faster or have a smaller implementation footprint than comparable block ciphers. They operate more naturally on data of short, variable or unknown length. Finally, the keystream generation is completely independent of the plaintext data, and so it may be computed in parallel with or in advance of the data stream. In general, it is also useful for system designers to have a reasonable selection of different encryption algorithms to choose between, as this makes it possible both to select precise performance tradeoffs suitable for a specific application, and to avoid a cryptographic monoculture where everyone is

using AES and subsequently everyone gets in big trouble if future cryptanalysis reveals weaknesses in AES.

2 What is a secure stream cipher?

The usual starting assumption for attacks on stream cipher, is that the attacker has access to large amounts of keystream, generated under a number of different IVs which may (if necessary) be adaptively chosen by the attacker. In a sense this is a very generous setting; it is to be hoped that an actual real-world attack scenario will be (much) less bad. But it corresponds well with Kerckhoff's second principle: one always should assume that the enemy knows everything about the encryption system used, apart from the secret key itself. Conversely, if there are no attacks that can be applied even in this rather theoretical setting, there will surely not be any attacks in a more restricted (and possibly more realistic) situation.

There are two main criteria for the security of a stream cipher with a k-bit key. Firstly, the attacker should not be able to predict future keystream output by the cipher, whether this happens by recovering the secret key, recovering the internal state of the cipher at some point, or otherwise. The attacker can obviously do this by testing all possible secret keys, so the complexity of a brute force attack (requiring at most 2^k executions of the algorithm) gives a performance baseline to which any alleged attack should be compared.

Secondly, the attacker should not be able to distinguish keystream from random under the given usage limits for the cipher. While a distinguishing attack is certainly less serious than a full state recovery, it does indicate that the algorithm has some kind of undesirable internal structure. It should be emphasised that the existence of an "attack" of any kind on a stream cipher, does not mean that the attack is *practical* in any way. Rather, it implies that the algorithm is strictly worse than a hypothetical *ideal* stream cipher, where the only applicable attacks are generic attacks such as brute-forcing the key, and the keystream is truly indistinguishable from random when the key is unknown.

3 What is the current state of the art?

The most widely used stream cipher around is, by far, RC4. Designed by Ronald Rivest in 1987, it is extremely fast in software and can be implemented in just a few lines of code. The history of RC4 is actually quite interesting, as the cipher was originally considered a trade secret by RSA, and only became public knowledge after it was anonymously leaked on the Internet in 1994 [14]. Unfortunately, the age of RC4 is increasingly starting to show. For one, the cipher specification does not specify how to use an initialisation vector with the algorithm, which means that implementors have to be very careful about how they do this if they want to generate multiple keystreams based on the same long-term secret. More seriously, it was discovered in 2001 that the RC4 keystream exhibits various statistical biases that can be used to distinguish it form random and relate it to

the underlying secret key [10], and improved attacks along these lines have also been found since then. A usual technique to mitigate these attacks is to discard the first N bytes of the RC4 keystream; a typical value is $N = 1024$.

These issues imply that RC4 can not be considered (theoretically) secure by modern standards, as discussed in the previous section. Even though it may still be possible to use RC4 in a sufficiently secure way by a careful (and lucky) implementor, the WEP fiasco as well as the recent attack on WPA by Beck and Tews [2] show that the theoretical weaknesses of RC4 also lead to practical attacks on protocols in which RC4 is used. From an academic point of view, certainly, RC4 should not be used for new applications – even though it offers very attractive performance and ease of implementation from an engineering point of view, it is simply considered too risky and too difficult to get right.

Unfortunately, the state of other popular stream ciphers is no less dire. The A5/1 cipher and its variants used in GSM have a completely inadequate key length of 54 bits, and additional attacks have been found which are faster than brute force. Similarly, E0 has been broken by cryptanalysts. The European Union-based NESSIE project [12], which was aimed at evaluating the security of various cryptographic primitives, did not recommend any stream ciphers in their final report, because all the submitted algorithms were successfully attacked. Although there do exist various other stream ciphers around that are still unbroken (notably SNOW 2.0 [13], a tweaked variant of one of the NESSIE ciphers), none of them have really gained widespread acceptance and recognition outside the academic community.

The only popular, secure and widespread "stream cipher" that remains is, in the author's opinion, AES (or any other secure block cipher, but AES is after all the standard) operating in counter (CTR) mode. But this yields the obvious question: is it really not possible to design a secure, special-purpose stream cipher, which is more efficient than what you get by adapting a block cipher to the task?

4 What is the eSTREAM project?

The eSTREAM project [9] was launched as part of the EU-funded network of excellence ECRYPT [8], which ran from 2004 to 2008. This was partly in response to the dismal showing of the NESSIE stream ciphers, and had a stated project goal to identify "new stream ciphers that might become suitable for widespread adoption".

A call for primitives was put forward in the fall of 2004, and attracted 34 submissions from all over the world. The candidate algorithms were divided in two categories, or profiles, one for software-oriented algorithms, and one for ciphers suitable for hardware implementation. The design goals for the two profiles were somewhat different. Software candidates should offer a key size of at least 128 bits, and provide some significant advantage over the state of the art (i.e. AES-CTR) with respect to throughput. Candidates for the hardware profile should outperform AES in restricted environments with respect to relevant parameters

such as gate count, power consumption and speed, and provide a key size of at least 80 bits.

After three evaluation phases, the eSTREAM project ended in the spring of 2008 with a final portfolio of 8 "promising" ciphers, 4 in each profile. One of these, F-FCSR, was removed from the portfolio in September 2008 after new cryptanalytic results were found. It is important to remember that, unlike the previous AES competition and the current SHA-3 competition, the goal of eSTREAM was not to develop a new international standard for stream ciphers, but merely to act as a focus for academic interest, and attempt to identify the best candidates among the various designs. While the different eSTREAM algorithms are still quite new and untested and new weaknesses may yet be found, the portfolio can be considered to represent the current state of academic research on stream ciphers.

The eSTREAM portfolio consists, as of November 2008, of the following seven stream ciphers:

- HC-128 [15] (software), supporting 128-bit keys.
- Rabbit [6] (software), supporting 128-bit keys.
- Salsa20/12 [5] (software), supporting 128 and 256-bit keys.
- SOSEMANUK [3] (software), supporting 128-256 bit keys.
- Trivium [7] (hardware), supporting 80-bit keys.
- Grain v1 [11] (hardware), supporting 80-bit keys.
- MICKEY v2 [1] (hardware), supporting 80-bit keys.

In the accompanying lecture, the aim of the author is to give a lightning tour of the eSTREAM portfolio ciphers, emphasising their respective strengths and weaknesses, and examining how the different algorithms are constructed.

References

1. S. Babbage and M. Dodd. The MICKEY stream ciphers. *Lecture Notes in Computer Science*, 4986:191–209, 2008. `http://www.ecrypt.eu.org/stream/mickeypf.html`.
2. M. Beck and E. Tews. Practical attacks against WEP and WPA, 2008. `http://dl.aircrack-ng.org/breakingwepandwpa.pdf`.
3. C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert. SOSEMANUK, a fast software-oriented stream cipher. *Lecture Notes in Computer Science*, 4986:98–118, 2008. `http://www.ecrypt.eu.org/stream/sosemanukpf.html`.
4. D. Bernstein. Notes on the ECRYPT Stream Cipher project (eSTREAM). `http://cr.yp.to/streamciphers.html`.
5. D. Bernstein. The Salsa20 family of stream ciphers. *Lecture Notes in Computer Science*, 4986:84–97, 2008. `http://www.ecrypt.eu.org/stream/salsa20pf.html`.
6. M. Boesgaard, M. Vesterager, and E. Zenner. The Rabbit stream cipher. *Lecture Notes in Computer Science*, 4986:69–83, 2008. `http://www.ecrypt.eu.org/stream/rabbitpf.html`.
7. C. de Cannière and B. Preneel. TRIVIUM. *Lecture Notes in Computer Science*, 4986:244–266, 2008. `http://www.ecrypt.eu.org/stream/triviumpf.html`.

8. ECRYPT Network of Excellence in Cryptology. http://www.ecrypt.eu.org/.
9. The eSTREAM project. http://www.ecrypt.eu.org/stream/.
10. S. Fluhrer, I. Mantin, and V. Shoup. Weaknesses in the key scheduling algorithm of RC4. In *Proceedings of SAC 2001*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer–Verlag, 2001.
11. M. Hell, T. Johansson, A. Maximov, and W. Meier. The Grain family of stream ciphers. *Lecture Notes in Computer Science*, 4986:179–190, 2008. http://www.ecrypt.eu.org/stream/grainpf.html.
12. New European Schemes for Signatures, Integrity and Encryption. http://www.cosic.esat.kuleuven.be/nessie/.
13. T. Johansson P. Ekdahl. A new version of the stream cipher SNOW. In *Proceedings of SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 47–61. Springer–Verlag, 2002. http://www.it.lth.se/cryptology/snow/.
14. RC4 algorithm revealed. http://groups.google.com/group/sci.crypt/msg/10a300c9d21afca0.
15. H. Wu. The stream cipher HC-128. *Lecture Notes in Computer Science*, 4986:39–47, 2008. http://www.ecrypt.eu.org/stream/hcpf.html.