# Why IV Setup for Stream Ciphers is Difficult

Erik Zenner
Technical University of Denmark*
e.zenner@mat.dtu.dk

March 14, 2007

**Abstract**

In recent years, the initialization vector (IV) setup has proven to be the most vulnerable point when designing secure stream ciphers. In this paper, we take a look at possible reasons why this is the case, identifying numerous open research problems in cryptography.

## 1 Introduction

**Motivation:** Traditionally, a cipher was defined as an algorithm that transforms a message into a ciphertext under the control of a secret key (see, e.g., [20, 25]). Stream ciphers were no exception to this rule. In recent years, however, an additional input parameter has gained in importance: the initialization vector (IV). As opposed to the key, the IV is public, and a new IV is used for each new message. The IV has two main uses: It provides randomized encryption, and it helps in synchronizing communication between sender and receiver.

As it turns out, however, incorporating an IV into a stream cipher is not an easy task. An important reason for this is that the IV is an input parameter which is partially under the control of the adversary [23]. Giving the adversary control over an input parameter significantly increases the range of attacks he has available. As an example, think of a block cipher adversary who is restricted to a known-plaintext scenario, as opposed to a block cipher adversary who also has chosen plaintext or chosen ciphertext at his disposal.

As a consequence, attacks against the IV setup of stream ciphers have been very successful recently:

- Out of the 34 stream ciphers submitted to the eStream project [12] in May 2005, 10 were broken due to problems in the IV setup by December 2006 (see, e.g., [7, 31, 32]). This is more than 25% of all submissions.

*This paper accompanies a talk given at the Dagstuhl Seminar on "Symmetric Cryptography" on January 11, 2007. At the time of the talk, the author's affiliation was Cryptico A/S, Copenhagen.

- Examples for seemingly strong academical designs that were broken due to problems with the IV setup are Turing [17] and Helix [21].
- Notable examples for fielded stream ciphers that were broken due to problems with the IV setup are the GSM standard [11] and some implementations of the WEP standard [14].

Thus, it seems that we need a better understanding of what the IV setup is supposed to achieve, and how those goals can be met.

**Engineering approach:** When designing an object (not necessarily a cryptographic one), a sensible approach is to first define its intended properties, then to look for prior knowledge on similar objects, and only then to build it. If we apply this approach to the design of an IV setup for stream ciphers, then we have to start by asking ourselves what it is that we want to achieve (security requirements), continue by learning as much as possible from past experiences (construction principles), and then start the actual design process.

However, as mentioned above, our knowledge about an IV setup as such is currently limited. If we do not want to start from scratch, it makes sense to look into related fields within cryptography. An obvious starting point is to consider the security requirements and construction principles for related objects. If we find anything useful there, we have to adapt our findings to the stream cipher setting. In order to do this, we have to be aware of the security requirements for the stream cipher (as a whole, as opposed to security requirements for the IV setup only). And finally, in order to define properties for a stream cipher, we have to agree first what a stream cipher is.

The rest of this paper deals with those questions, in reverse order.

**Purpose:** This paper is not about the outcome, but about the start of a research project. As a consequence, its purpose is to collect initial ideas and research questions. The reader is welcome to work on the questions asked. The author will be grateful for any pointers, both to prior art that is missing in this paper and to new research done after the paper was published.

# 2 What is a stream cipher?

## 2.1 Universal Secure Encryption (USE) Schemes

**Introduction:** Before we consider an IV setup for stream ciphers, we first have to be clear about what a stream cipher is. In fact, it turns out that definitions given by both researchers and practitioners are misleading. Let us start by quoting the (anonymized) chief security architect of one of the world's largest companies. The following statement was made in e-mail communication in Summer 2005:

> "Stream ciphers are always faster than block ciphers [...] but are considered much easier to break because of problems with the key handling and the pseudo nature of the number generators."

Statements like those are wide-spread amongst practitioners and are motivated mainly by the security breaks against fielded systems in e.g. WEP and GSM. But are they correct? As cryptographers, we observe that this statement actually compares apples with oranges, since a stream cipher is used for encryption, while a (raw) block cipher is not. Block ciphers are always used in a mode of operation, and those require key and IV handling, too.

In fact, practitioners do not care much about the internals of an encryption algorithm. They do not care about the differences between block and stream ciphers. They might not even be aware that there is a distinction between a "cipher" and a "secure cipher". All they care about is a universal encryption algorithm that can process all kinds of input and that encrypts and decrypts in a secure way. This leads us to the following, informal definition of what really is required for practical use:

**Definition 1** *A **universal secure encryption (USE) scheme** is a cipher with the additional properties that (a) it can process messages of arbitrary bit length and (b) it is secure.*

Note that the definition of a USE scheme is not identical to the standard textbook definition of a cipher or even a secure cipher. In particular, a block cipher can not be a USE scheme since it does not process messages of arbitrary length.

**Bellare and Rogaway's definition:** Searching the cryptographic literature for a matching definition, the one given by Bellare and Rogaway in [3], p. 93, comes close. Here, a **symmetric encryption scheme** consists of the following components:

- A randomized **key generation algorithm**.
- A randomized or stateful **encryption algorithm** processing messages of arbitrary length.
- A deterministic **decryption algorithm**.

Such a symmetric encryption scheme is considered secure ([3], pp. 102-103) if no adversary with realistic restrictions for time and hardware can win the following game with a non-negligible advantage (IND-CPA security):

- The adversary chooses pairs of messages from $\{0,1\}^*$ (each pair having equal length) and sends them to the encryption oracle.
- The oracle is either an L- or an R-oracle. An L-oracle encrypts the first, an R-oracle the second message.

- The adversary wins if he can tell whether he communicates with an L- or an R-oracle.

Given these notions of symmetric encryption scheme and security, the following corollary can be proven ([3], pp. 107-108):

**Corollary 1** *Any deterministic, stateless symmetric encryption scheme is insecure.*

Since a secure symmetric encryption scheme is identical to our notion of a USE scheme, we conclude that all USE schemes require initialization vectors - either for randomization or to remember the last state. A further consequence is that block ciphers in ECB mode are not USE schemes. This begs the question of whether or not stream ciphers are USE schemes. But in order to give a meaningful answer to this question, we first have to give a definition of a stream cipher.

## 2.2 Stream cipher definitions

Even though stream ciphers are considered as the opposite of block ciphers, they do nonetheless process plaintext and ciphertext in blocks of $b$ bit. The classical (hardware-based) stream ciphers operated on individual bits, meaning $b = 1$, while modern (software-based) stream ciphers often use larger blocks like $b = 8$, $b = 32$ or even $b = 256$.

**Narrow definition:** The following definition covers a very narrow understanding of a stream cipher that is frequently used by practitioners.

**Definition 2** *Let $b$ be the block length of the stream cipher. Let $\oplus$ denote bitwise exclusive-or operation (aka. addition over GF($2^b$)). Then a stream cipher works as follows:*

- *A pseudo-random generator (PRG) expands a short key and initialization vector (IV) into a long pseudo-random keystream, consisting of $b$-bit words $s_0, s_1, \ldots$.*
- *The plaintext is subdivided into $b$-bit blocks $m_0, m_1, \ldots, m_{l-1}$. It is encrypted by computing $b$-bit ciphertext blocks $c_i = m_i \oplus s_i$, for $i = 0, \ldots, l-1$.*
- *Analoguously, the ciphertext is decrypted by reconstructing the message as $m_i = c_i \oplus s_i$, for $i = 0, \ldots, l-1$.*

The obvious problem with this definition is that it does not cover some of the designs that are also considered as stream ciphers, as for example self-synchronizing stream ciphers (like the eStream candidate Moscito/Moustique [9]) or stream ciphers with authentication (like the eStream candidate Phelix [29]), where the inner state of the PRG depends not only on the key and IV, but also on the plaintext. Thus, we either have to consider these ciphers as not being stream ciphers, or we have to use a broader stream cipher definition.

**Broad definition:** A broader definition was given by Rueppel [24] and is used, e.g., in the "Handbook of Applied Cryptography" [20]. The Handbook describes the difference between block and stream ciphers as follows:

> **Stream ciphers** [...] encrypt individual characters [...] of a plaintext message one at a time, using an encryption transformation which varies with time. By contrast, **block ciphers** [...] tend to simultaneously encrypt groups of characters of a plaintext message using a fixed encryption transformation.

It is obvious that the difference between encrypting "individual characters" and "groups of characters" is meaningless in practice, since it depends completely on how a character is defined. In practice, both stream and block ciphers operate on $b$-bit blocks, with $b$ depending on the cipher itself and typically being in the range between 64 and 256 bit for block ciphers and 1 and 256 bit for stream ciphers.

The remaining characteristic of a stream cipher according to Rueppel is thus that it transforms plaintext blocks into ciphertext blocks in a *time-varying* fashion. This means that even if the same plaintext block occurs twice in a message, the corresponding ciphertext blocks will not necessarily be the same. With other words, there must exist a method of preserving an inner state between the processing of two plaintext blocks, either by re-initializing the state with an IV or by storing the state itself.

Remembering our observations on USE schemes, we find that it was a necessary condition for USE schems to use either randomization or to store the inner state. While this does not imply that all stream ciphers are USE schemes, it does imply that all USE schemes are stream ciphers according to Rueppel's definition!

**Conclusion:** It seems that we can not find a useful definition for a stream cipher in the literature. The narrow definition does not cover all designs currently denoted as stream ciphers in the cryptographic community, while the broad definition covers all secure encryption schemes.

It was pointed out by Bart Preneel during the Dagstuhl seminar talk that a rigorous definition might not be needed in practice. As an example, he stated that program committees typically do not have a problem with placing submissions into the "stream cipher" or the "block cipher" category. But this placement seems to be based mainly on intuition, and it can be ambivalent[1]. In addition, placing ciphers into conference sessions is not the only purpose of such a categorization.

Practitioners get confused by unclear nomenclature, since they use such categories to pre-select which cipher to use. If some algorithms in a category

[1] As an example, does a contribution on counter mode encryption belong into the block cipher or into the stream cipher category? If it is placed in the block cipher category, then a cipher like Salsa20 [5] should be placed in the hash function category for the same reasons. On the other hand, if it is placed in the stream cipher category, then what about CBC mode encryption?

fail, then people without in-depth knowledge will consider the whole category to be unsuitable to their purposes. And if the labels are chosen sloppily, then they might end up making the wrong choice. This is the case with block and stream ciphers: The Chief Security Architect mentioned in Subsection 2.1 heard bad things about stream ciphers, so she chose to use block ciphers instead, which does not make her choice one bit more secure. Quite the opposite: If she uses a block cipher in counter mode, she ends up having exactly the same IV problem that she wanted to avoid before. She is not even aware that depending on the interpretation of the term "stream cipher", some or all block cipher modes of operation *are* stream ciphers, making even a technically more correct statement like "stream ciphers are less secure than block cipher based encryption schemes" meaningless.

# 3 Stream cipher model

## 3.1 Stream cipher definition

**Additive stream cipher:** Returning to our original intention of designing an IV setup for a stream cipher, we choose to restrict our concept of a stream cipher as much as possible in order to make meaningful statements. Thus, we will subsequently use the narrow definition, which is sometimes denoted as additive stream cipher. In the following, our stream cipher consists of a **pseudo-random generator (PRG)** which expands key and IV into a keystream. This keystream is then xored to the message in order to encrypt, and to the ciphertext in order to decrypt. The principle is described in Figure 1.

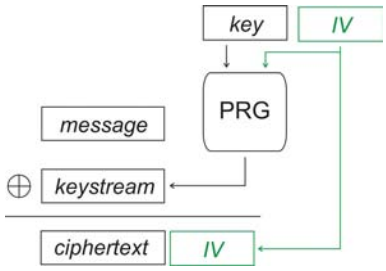
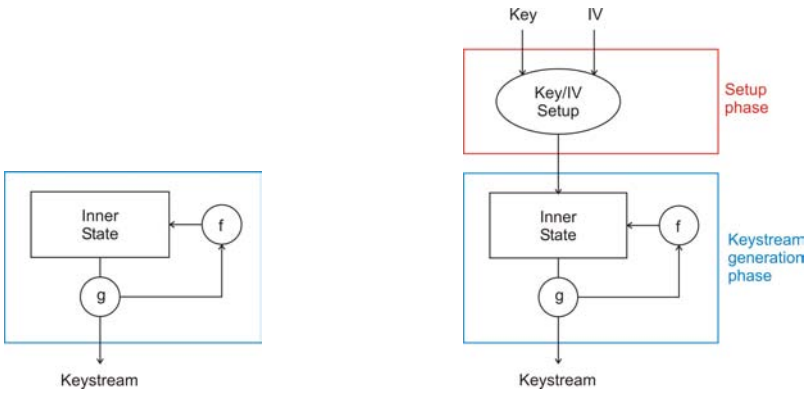



Figure 1: PRG-based stream cipher

**Construction principle:** Pseudo-random generators have been discussed in cryptographic literature in many years, and they are relatively well understood. Traditionally, the initial inner state of the PRG was considered as the key. A function $f$ updates the inner state between two iterations, and a function $g$ generates a number of output bits from the inner state. The illustration in part (a) of Figure 2 illustrates the principle.

The problem with this kind of generator is that in order to serve as stream cipher, it lacks key/IV setup. Thus, we need an additional component that mixes the key and initialization vector into the initial state of the PRG, as illustrated in part (b) of Figure 2.




(a) Traditional PRG (b) PRG with key/IV setup

Figure 2: Pseudo-random generator for stream ciphers

It is the new "key/IV setup" component that gives rise to the problems with modern stream ciphers. We know that the key and IV have to be mixed into the inner state, but how? Both security requirements and construction principles are not well understood. Thus, in Section 4, we will look at related building blocks in order to find inspiration.

## 3.2 Stream cipher security

**Security definition:** Before we can identify meaningful security criteria for the key/IV setup, we have to review the security requirements for a stream cipher as a whole. To this purpose, we use the IND-CPA security definition that was described in Subsection 2.1. In addition, we follow the recommendation by Rogaway [23] who proposes to explicitly give the adversary control over the IV. This means that in addition to choosing messages (which is not an advantage in our narrow stream cipher model), the adversary also is allowed to choose the IV that is used for the encryption, as long as he does not request the same IV twice (nonce-respecting adversary).

**Possible extensions:** A number of possible extensions to this model have implicitly been proposed in recent years:

- In 2005, Hong and Sarkar [15] proposed a multiple-key attack model. In this model, the adversary attacks a system where the same encryption algorithm is used under several keys. He is already considered successful if he can achieve his goal for any one key that is being used. In addition, Hong

and Sarkar presume that the adversary has unlimited pre-computation time at his disposal and that the only thing that prevents him from pre-computing the whole key/IV space is lack of memory. There is currently no unanimous agreement regarding whether or not this model should be considered valid or not.

- In 2006, Wu and Preneel [30] proposed an attack against Phelix that was based on re-using the initialization vector. With other words, the adversary in this model is no longer nonce-respecting. This attack model was met by strong opposition by some cryptographers, notably Bernstein [13, 19], who state that if the users do not follow precautions necessary to use an algorithm, then the algorithm can hardly be considered broken. As an example, all additive stream ciphers are broken under the above security definition if the adversary is not nonce-respecting.

Since these new attack models are disputed, we do not take them into consideration here, but name them merely to illustrate that the choice we made was just one out of several possible ones, and that no security model exists that the cryptographic community agrees with unanimously.

# 4 Key/IV setup for stream ciphers

## 4.1 Modelling the key/IV setup

After clarifying the nature of a stream cipher and the security requirements, we can return to our main goal of making statements about the IV setup of stream ciphers. We start by repeating the question about the security requirements for a key/IV setup.

From our definitions for a stream cipher and its security, we can informally conclude that a stream cipher is secure if the adversary is unable to win the following (informal) game:

- He sends pairs (IV, $n$) to an oracle and receives a bitstream of length $n$ in reply.
- He wins if he can tell whether the bitstreams were generated by the stream cipher or whether they are random.

The stream cipher is considered secure if no adversary (with realistic resource restrictions) has a non-negligible chance of winning this game.

Let us assume now that the PRG part of the stream cipher is secure (and has a sufficient security level). In this case, a sufficient condition for the overall security is that the output of the key / IV setup (i.e., the initial state) can not be distinguished from random. This requirement forms a starting point in looking for design principles for a key/IV setup. In the following, we try to borrow inspiration from related cryptographic primitives and look for suitable building blocks, their definitions, construction principles, and efficiency.

## 4.2 First idea: Hash function

**Intuition:** Informally, a hash function maps an input of arbitrary length onto an output of fixed length in such a way that the output "looks random". In fact, if the output of a hash function can not be distinguished from random, then this hash function would solve our problem.

**Definition:** The problem is that no known formalization describes the "looks random" property correctly. Originally, hash functions were supposed to provide the properties of one-wayness, second preimage resistance, and collision resistance. None of those properties corresponds to the requirement that the output should be indistinguishable from random.

Hash functions are sometimes also used as instantiations of random oracles [2]. This model could be used for our purposes, but even though it might be correct in practice, it can be shown in theory that no instantiations of random oracles can exist [6]. In addition, usefulness as a random oracle is not a design goal of hash functions, but rather a byproduct. Thus, there is no unanimous agreement for the claim that hash functions produce output that is indistinguishable from random.

**Construction principles:** Most modern hash functions are based on the so-called Merkle-Damgaard (MD) paradigm (see e.g. [25], pp. 131-137). However, the recent attacks on the most prominent hash functions like MD5 and SHA-1 [28, 27] as well as generic attacks against the MD construction [16] have shown that this paradigm has its weaknesses. In fact, at the NIST workshop on hash functions in 2006, leading experts in the field agreed that our understanding of construction principles for hash functions is incomplete [22]. Major research in this area is expected, meaning that it might in fact be dangerous to rely on hash functions for constructing our key/IV setup at the moment[2].

**Efficiency:** In addition, general-purpose hash functions have the disadvantage that they might be more inefficient than required for our purposes. The reason for this is twofold:

- As opposed to a key/IV setup, a hash function does not require a key. Since all of their input is known to the adversary, the resulting construction has to be stronger than a similar construction using a key. This implies that it might be possible to build a dedicated key/IV setup that is more efficient than modern hash functions.
- While a key/IV setup typically processes less than 1000 bit of input, a hash function is designed to process inputs of arbitrary length. Again,

[2]For completeness sake, we mention that most attacks currently proposed against hash functions might in fact have no relevance in a scenario where the hash function is used as key/IV setup. However, this only serves to show that hash functions try to achieve security properties that are quite different from those of a key/IV setup.

this means that dedicated hash functions have an overhead for handling long messages and for preventing length-extension attacks that might not be required in our case.

Thus, the efficiency of a hash function might be suboptimal in our case, and dedicated constructions might be more efficient.

## 4.3 Second idea: Key derivation function

**Intuition:** A key derivation function (KDF) takes an input that has a secret and a public part and generates an output that can be used as a key (e.g. for a PRG). Since this output should be indistinguishable from random, a KDF might solve our problem.

**Definition:** As it turns out, our understanding of what a KDF actually is is even more vague than for a hash function. A long discussion on CFRG mailing list [18] in 2005 showed that most researchers have an intuition of what a KDF is, but that a definite definition is lacking. It seems likely, though, that if the public and the secret part of the KDF input are cleanly separated[3], then the definition might be identical to that of a secure pseudo-random function (see below).

**Construction principles:** There are only few designs for dedicated KDFs. In fact, most KDFs in the literature (see e.g. [1] and the references contained therein) are built on hash functions or block ciphers, meaning that they are not sufficiently efficient for use with fast or compact stream ciphers.

## 4.4 Third idea: Pseudo-random function

**Intuition:** Intuitively, a pseudo-random function (PRF) maps a public input under a secret key onto an output that "looks random". Again, this might be a solution to our problem.

**Definition:** As opposed to hash functions and KDFs, PRFs have a well-understood security definition (see e.g. [3], pp. 64-67). This definition contains all the components that we are interested in: A public input (the IV), a secret input (the key), and an output that can not be distinguished from random (the initial state). This means that we can solve the problem by modelling the key/IV setup as a PRF. What is more, as opposed to hash functions and KDFs, this is even the theoretically most accurate description of our problem.

[3] As opposed to, e.g., a KDF that takes as input the output of a Diffie-Hellman key exchange [10] (containing a certain redundancy which is considered as the "public" part [26]) and transforms it into a shorter value with full entropy.

**Construction principles:** The main problem is that there are almost no dedicated PRFs proposed in the literature[4] In principle, message authentication codes (MACs) can be used, but there is a subtle theoretical difference between a MAC and a PRF (see e.g. [3], pp. 167-168), and the output lengths might not be what we are looking for. Nonetheless, the design principles of dedicated PRFs and MACs currently seem to be the best starting point when we want to learn something about construction principles for key/IV setup.

**Efficiency:** As with hash functions, existing PRFs might be more complicated than required for a key/IV setup. This can be seen from the fact that the majority of existing key/IV setup functions (even those that are unbroken) are too weak for a PRF. This indicates that random indistinguishability of the initial state is a sufficient, but might not be a necessary criterion for a key/IV setup function.

### 4.5 Outlook

Thus, of the related primitives considered, the pseudo-random function (PRF) seems to be most suited as inspiration for a key/IV setup. However, it also seems that a really efficient key/IV setup can be built to be more efficient than a PRF. But in order to do this, the designer will have to take the properties of the PRG into account - the stronger the PRG, the weaker the key/IV setup can be allowed to be. This was also pointed out by participants of the seminar, notably Henri Gilbert and Greg Rose. As an example, the key/IV setup of the block cipher counter mode consists in just writing the key and IV into the correct locations in memory. Of course, the resulting initial state can easily be distinguished from random, but nonetheless, the block cipher counter mode is considered secure. The same holds for the Salsa20 stream cipher [5], which uses a hash function in counter mode. This dependency between the strength of the key/IV setup and the PRG makes design and analysis of an efficient and secure stream cipher considerably more complicated.

## 5 Conclusions

Trying to design a key/IV setup for stream ciphers has taken us into a number of realms of cryptography that are not well-understood. As it turns out:

- There is no useful definition of a stream cipher.
- Even for a narrow definition of a stream cipher, cryptographers do not agree on what constitutes a valid attack.
- Even for a narrow definition of an attack, it is unclear what the key/IV setup is supposed to achieve.

[4] At the Dagstuhl seminar, Joan Daemen pointed out that Panama [8] was an example for such a dedicated PRF.

- Related building blocks that could be considered as candidate key/IV setup or as inspiration (namely hash functions, KDFs, and PRFs) have at least one of the following problems:
  - The security definition is unclear.
  - Wide-spread constructions are again based on other primitives (notably block ciphers); no dedicated constructions exist.
  - Existing dedicated constructions are insecure.

Considering that not only key/IV setup of stream ciphers, but also almost all related areas of research are not as well understood as one might hope, it is not that surprising that designing a stream cipher IV setup has proven to be difficult in the past.

## Acknowledgements

The author wishes to thank all participants in the Dagstuhl seminar on Symmetric Cryptography for the lively discussions and many pointers.

The author also wishes to mention a paper by Berbain and Gilbert that will be presented at FSE 2007 and that touches on some of the issues mentioned in this article. This paper [4] was not publicly available by the time of the Dagstuhl seminar yet.

## References

[1] C. Adams, G. Kramer, S. Mister, and R. Zuccherato. On the security of key derivation functions. In K. Zhang and Y. Zheng, editors, *Proc. ISC 2004*, volume 3225 of *LNCS*, pages 134–145. Springer, 2004.

[2] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proc. of First Annual Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[3] M. Bellare and P. Rogaway. Introduction to modern cryptography. http://www.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf, May 2005.

[4] C. Berbain and H. Gilbert. On the security of IV dependent stream ciphers. In *Proc. FSE 2007*, LNCS. Springer. to appear.

[5] D. Bernstein. Salsa20 specification. http://www.ecrypt.eu.org/stream/salsa20.html, 2005.

[6] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proc. 30th STOC*, pages 209–218. ACM, 1998.

[7] C. Cid, H. Gilbert, and T. Johansson. Cryptanalysis of Pomaranch. *IEE Proc. Information Security*, 153(2):51–53, June 2006.

[8] J. Daemen and C. Clapp. Fast hashing and stream encryption with Panama. In S. Vaudenay, editor, *Proc. FSE '98*, volume 1372 of *LNCS*, pages 60–74. Springer, 1998.

[9] J. Daemen and P. Kitsos. The self-synchronizing stream cipher Moustique.
http://www.ecrypt.eu.org/stream/ciphers/mosquito/mosquito_p2.pdf, 2006.

[10] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, IT-22(6):644–654, 1976.

[11] P. Ekdahl and T. Johansson. Another attack on A5/1. *IEEE Trans. Information Theory*, 49(1):284–289, 2003.

[12] eStream - the ECRYPT stream cipher project.
http://www.ecrypt.eu.org/stream/.

[13] Phorum eStream. Key recovery attacks on Phelix.
http://www.ecrypt.eu.org/stream/phorum/read.php?1,883,921, 2006.

[14] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of RC4. In A. Youssef S. Vaudenay, editor, *Proc. SAC 2001*, volume 2259 of *LNCS*, pages 1–24. Springer, 2001.

[15] J. Hong and P. Sarkar. New applications of time memory data tradeoffs. In B. Roy, editor, *Proc. Asiacrypt 2005*, volume 3788 of *LNCS*, pages 353–372. Springer, 2005.

[16] A. Joux. Multicollisions in iterated hash functions - applications to cascaded constructions. In M. Franklin, editor, *Proc. Crypto 2004*, volume 3152 of *LNCS*, pages 306–316, Berlin, 2004.

[17] A. Joux and F. Muller. A chosen IV attack against Turing. In M. Matsui and R. Zuccherato, editors, *Proc. SAC 2003*, volume 3006 of *LNCS*, pages 194–207. Springer, 2004.

[18] Cfrg Mailing List. Fwd: Hash-based key derivation.
http://www1.ietf.org/mail-archive/web/cfrg/, October 2005.

[19] Cfrg Mailing List. Consequences of nonce reuse.
http://www1.ietf.org/mail-archive/web/cfrg/, January 2007.

[20] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[21] F. Muller. Differential attacks against the Helix stream cipher. In B. Roy and W. Meier, editors, *Proc. FSE 2004*, volume 3017 of *LNCS*, pages 94–108. Springer, 2004.

[22] J. Nechvatal and S. Chang. Workshop record: The second cryptographic hash workshop. available from: http://www.csrc.nist.gov/pki/HashWorkshop/, August 2006.

[23] P. Rogaway. Nonce-based symmetric encryption. In B. Roy and W. Meier, editors, *Proc. FSE 2004*, volume 3017 of *LNCS*, pages 348–359. Springer, 2004.

[24] R. Rueppel. *Analysis and Design of Stream Ciphers*. Springer, 1986.

[25] D. Stinson. *Cryptography - Theory and Practice*. Chapman and Hall, 3rd edition, 2006.

[26] C. Waldvogel and J. Massey. The probability distribution of the Diffie-Hellman key. In J. Seberry and Y. Zheng, editors, *Proc. Asiacrypt '92*, volume 718 of *LNCS*, pages 492–504, Berlin, 1993. Springer.

[27] X. Wang, Y. Yin, and H. Yu. Finding collisions in the full SHA-1. In V. Shoup, editor, *Proc. Crypto 2005*, volume 3621 of *LNCS*, pages 17–36, Berlin, 2005. Springer.

[28] X. Wang and H. Yu. How to break MD5 and other hash functions. In R. Cramer, editor, *Proc. Eurocrypt 2005*, volume 3494 of *LNCS*, pages 36–57, Berlin, 2005. Springer.

[29] D. Whiting, B. Schneier, S. Lucks, and F. Muller. Phelix - fast encryption and authentication in a single cryptographic primitive. http://www.ecrypt.eu.org/stream/ciphers/phelix/phelix.pdf, 2005.

[30] H. Wu and B. Preneel. Differential-linear attacks against the stream cipher Phelix. In *Proc. FSE 2007*, LNCS. Springer. to appear.

[31] H. Wu and B. Preneel. Cryptanalysis of the stream cipher DECIM. In M. Robshaw, editor, *Proc. FSE 2006*, volume 4047 of *LNCS*, pages 30–40. Springer, 2006.

[32] H. Wu and B. Preneel. Resynchronization attacks on WG and LEX. In M. Robshaw, editor, *Proc. FSE 2006*, volume 4047 of *LNCS*, pages 422–432. Springer, 2006.