Vector Stream Cipher Instant Key Recovery

Sean O'Neil
Synaptic Laboratories Ltd

Created: 11 September 2004
Public release: 27 January 2006

Abstract: Vector Stream Cipher (VSC) is a stream cipher designed by ChaosWare and patented by NICT (formerly CRL), Japanese patents 3030341 and 3455758, US patent 6,668,265. VSC is recommended by the Softbank Technology Corporation for use in high bandwidth and high security applications. In this paper we present a practical attack instantly recovering the entire secret key of the high-speed single-round VSC variants with only 4 known subsequent plaintext blocks showing how all single-round VSC variants can be trivially broken due to their simple algebraic nature. The algorithm presented in this paper cannot break the 8-round VSC, but it can be easily adapted to any particular high-speed single-round VSC variant and extended to break some of the multiple-round VSC variants with very little effort and it may help accelerate other attacks.

1. Introduction

Vector Stream Cipher (VSC) is a stream cipher designed by ChaosWare and patented by NICT (formerly Communications Research Laboratory, CRL), Japanese patents 3030341 and 3455758, US patent 6,668,265. VSC is claimed to be based on Chaos theory and is recommended by the Softbank Technology Corporation for use in high bandwidth and high security applications, such as HDTV. VSC has also received the 5th LSI IP Design Award by the ST Microelectronics in Japan in 2003. An 8-round variant of the VSC Type 3 was "independently" evaluated by the SBT Corporation themselves reporting it as "*an independent 3rd party evaluation facility*" and "*confirming the encryption technique security*", while the single-round variant used in all high-speed hardware applications has never been evaluated.

VSC is claimed to be the fastest stream cipher in the world encrypting data streams at 25Gbps. Hardware implementation estimates included in Appendix A demonstrate that it is impossible for an 8-round variant of the VSC to achieve such high performance and that the figures provided by the ChaosWare match performance of the single-round variant. We have also received a verbal confirmation by ChaosWare that they offer a trade-off between speed and security by providing a single-round variant for high-speed applications.

The VSC key recovery process presented in this paper is not a theoretical attack and is not designed to break any particular hardware implementation of VSC since none are available to the authors of this paper to date. It only demonstrates the underlying weakness of the algebraic structure of the VSC according to all the available descriptions and software implementations. The authors have informed ChaosWare of the attack, but no response has been received. The C source code for the VSC key recovery process presented in this paper is available from Synaptic Laboratories upon request.

2. The VSC Algorithm

VSC state consists of w 32-bit blocks $x_0..x_{w-1}$ divided on two halves. The first half is initialised with a $w/2*32$-bit key, the second half is initialised with an IV of the same size. ChaosWare suggests 128, 256, 512 and 1024-bit key sizes labelling those variants as VSC-128, VSC-256, VSC-512 and VSC-1024 for w = 8, 16, 32 and 64 respectively. This algorithm assumes a zero-knowledge initial state.

One round of VSC executes a block of parallel [non-linear] multiplications modulo 2^{32} and one 5-bit rotation operation of the entire block. The exact number of bits used for rotation does not make much difference to the attacker and is named s. There are three different variants of VSC which only differ from each other by the order in which multiplication and rotation operations are executed:

2.1. **VSC Type 1:**

> $c = 4$;
> for each $i = [0..w\text{-}1]$ execute { $y_i = x_i (2x_i + 1 + c)$, $c = x_i$ AND NOT 3; }
> rotate $y_{0}..y_{w-1}$ left by s bits as a single w*32-bit wide block;

2.2. **VSC Type 2:**

> $c = 4$;
> rotate $x_{0}..x_{w-1}$ left by s bits as a single w*32-bit wide block;
> for each $i = [0..w\text{-}1]$ execute { $y_i = x_i (2x_i + 1 + c)$, $c = x_i$ AND NOT 3; }

2.3. **VSC Type 3:**

> for each $i = [0..w\text{-}1]$ execute { $c = x_{p(i)}$ AND NOT 3, $y_i = x_i (2x_i + 1 + c)$; }
> rotate $y_{0}..y_{w-1}$ left by s bits as a single w*32-bit wide block;

The new cipher state y replaces the old cipher state x and $x_{\underline{w/2}}..x_{w-1}$ values are released as the cipher output.

Only VSC Type 3 is published by ChaosWare, therefore this paper will only focus on this variant of the cipher. *Note: secret keys for Type 1 and Type 2 VSC ciphers can be trivially recovered by this algorithm as well with only minor alterations and 4-5 more required plaintext blocks while using only one 32-bit word from each block.*

Note: the $p(i)$ permutation used in the VSC Type 3 multiplication layer is only known for the VSC-128, it's {5,4,6,7,2,3,0,1}.

3. Multiplicative inverse mod 2^{32}

Multiplication mod 2^{32} by odd numbers is becoming more and more popular as a cryptographic primitive with modern processors being able to perform it in two clock cycles, practically as quickly as a bitwise rotation operation. It offers obvious benefits as a cryptographic primitive with its high polynomial degree of the relationship between the most significant bits of the output and most bits of the input, its fast avalanche of change in the least significant bits of input, and its reverse operation being the same. To reverse $y = x * m$ (assuming that m is known and is odd) one must simply multiply the result y by the multiplicative inverse of m mod 2^{32}:

$$x = y * \text{inv32}(m) \qquad (3.1)$$

Multiplicative inverse mod 2^{32} in the above formula can be calculated very quickly as follows:

```
// C source code
unsigned long inv32 (const unsigned long x)
{
    unsigned long a = 0, c = (x >> 1) + 1, d = x - 1, i;
    for (i = 1; i; i <<= 1) if (d & 1) d = (d >> 1) + c, a |= i; else d >>= 1;
    return a + 1;
}
```

A more detailed description of calculation of the multiplicative inverse mod 2^n can be found in literature. Easily calculated multiplicative inverse mod 2^{32} is also in the core of reversing and breaking LCG and similar generators that rely on multiplication mod 2^{32}.

Another obvious advantage of multiplication mod 2^{32} as a cryptographic primitive for use in modern 32-bit processors is that in some cases multiplicative inverses can be calculated off-line thus making decryption process completely identical to encryption.

Of course cryptographers must also keep in mind that multiplication discriminates very heavily between the most significant bits that are affected by every single bit lower than them and the least significant bits that are not affected by any of the bits positioned higher than them. Recovery of all the missing bits in the following process is trivial due to this imbalance. Multiplication, addition and rotation operations also leave highly accurate 'Mod n' distinguishers that make systems built without use of XOR operations vulnerable to the 'Mod n' attacks [3].

4. Division mod 2^{32}

In order to find $x_{p(i)}$ knowing y_i and x_i from two subsequent outputs we must first reverse the VSC formula $y_i = x_i (2x_i + 1 + c)$ finding c for arbitrary values of x_i and y_i. Note that x_i in this formula is multiplied by an odd number. If x_i is also odd, the above formula can be reversed directly, recovering the value of c unambiguously. The cases where x_i is an even number with the first n bits being zero will result in exactly n least significant bits of the result (y_i) being also zero. In these cases both x_i and y_i are first shifted to the right until they are both odd and then (3.1) is calculated for y_i and x_i, although in this case the n least significant bits of c [or the n most significant bits of $x_{p(i)}$] cannot be recovered directly from that formula.

Thus division mod 2^{32} is performed as follows:

```
// C source code
static unsigned long div32 (unsigned long x, unsigned long y)
{
    do if (y & 1) return x * inv32(y); else y >>= 1, x >>= 1; while (y);
    return 0;
}
```

The algorithm described below is limited to cases where no x_i has more than s least significant bits zero. For example, the probability of any given set of subsequent 4 outputs of VSC-128 containing no numbers with more than 5 least significant bits all zeroes is 75%. The algorithm present in this paper can be improved in a number of different ways to cover the remaining situations, but for the sake of simplicity of this publication we cover only the majority of cases where all the bits can be calculated trivially.

5. Recovering secret words

The attack described in this chapter assumes $w/2$ * 32-bit wide blocks of known plaintext/ciphertext available for 4 subsequent rounds r, r+1, r+2 and r+3. Words 0..$w/2$–1 of the cipher state on each round are assumed to be secret, while the words $w/2$..w–1 are known. For each i assuming values from $w/2$ to w–1, $p(i)$ assumes values between 0 and $w/2$–1, thus all the x^r_i are known output values on round r and all the $x^r_{p(i)}$ are secret values on round r.

5.1. The formula

The value of c from [2.3] can be calculated knowing two subsequent outputs:

$$x_{p(i)} = (\text{div32}\ (y_i, x_i) - 2x_i - 1)\ \text{AND NOT}\ 3 \qquad (5.1)$$

The left part of the above equation is $x_{p(i)}$ with its least significant two bits set to zero, therefore there are two more bits of $x_{p(i)}$ to be recovered. In cases where x_i is even, some of the most significant bits of $x_{p(i)}$ need to be recovered as well.

5.1.1. Recovering the two least significant bits of $x_{p(i)}$

Since the two least significant bits of c are always zero, multiplication of the two least significant bits of $x_{p(i)}$ by $2x_{p(i)} + 1$ in the following round would result in the two least significant bits of $y_{p(i)} = x_{p(i)} (2x_{p(i)} + 1)$ shifted to the left by s bits accepting values {0,3,2,1}

when the two least significant bits of $x_{p(i)}$ are {0,1,2,3} respectively. The simplest formula for this relationship is

$$y = (-x) \text{ AND } 3 \qquad (5.2)$$

Thus the two least significant bits of $x_{p(i)} = (-y_{p(i)} >> s) \text{ AND } 3$; and thus the complete formula for recovering $x_{p(i)}$ is:

$$x_{p(i)} = ((\text{div32}\ (y_i, x_i) - 2x_i - 1) \text{ AND NOT } 3) + ((-y_{p(i)} >> s) \text{ AND } 3) \qquad (5.3)$$

5.1.2. Recovering the s most significant bits of $y_{p(i)}$

When $y_{p(i)}$ value is recovered, some of its s most significant bits may be incorrect depending on the number of the least significant zeroes in x_i (or in y_i). Once all the 32–s least significant bits of $x_{p(i)}$ are known, the 32–s most significant bits of $y_{p(i)}$ can be recalculated with the complete formula from 2.3:

$$(x_{p(i)}(2x_{p(i)} + (x_{p(p(i))} \text{ AND NOT } 3) + 1)) << s \qquad (5.4)$$

In the above formula $p(p(i))$ assumes values from the same half of the state as i, hence all $x_{p(p(i))}$ are known outputs.

5.2. The order

First, let's determine in which order to recover the secret values of x "vertically", from one round to the next. The following is the list of (5.3) and (5.1) formulas recovering most bits of the secret values of x for the first three rounds. It must be noted that since $x^{r+3}_{p(i)}$ cannot be recovered without the $(r+4)^{\text{th}}$ round's outputs, the two least significant bits of $x^{r+2}_{p(i)}$ will not be known until all the secret values for at least one of the previous rounds are completely recovered, specifically the most significant bits of all the values of x^{r+2}.

$$\begin{aligned} x^{r+2}_{p(i)} &= ((\text{div32}\ (x^{r+3}_i, x^{r+2}_i) - 2x^{r+2}_i - 1) \text{ AND NOT } 3) \\ x^{r+1}_{p(i)} &= ((\text{div32}\ (x^{r+2}_i, x^{r+1}_i) - 2x^{r+1}_i - 1) \text{ AND NOT } 3) + ((-x^{r+2}_{p(i)} >> s) \text{ AND } 3) \\ x^{r}_{p(i)} &= ((\text{div32}\ (x^{r+1}_i, x^{r}_i) - 2x^{r}_i - 1) \text{ AND NOT } 3) + ((-x^{r+1}_{p(i)} >> s) \text{ AND } 3) \end{aligned} \qquad (5.5)$$

The values in (5.5) must be calculated beginning with x^{r+2}, and ending with x^r, since bits s and s+1 of x^{r+2} are required to recover the two least significant bits of x^{r+1}, and bits s and s+1 of x^{r+1} are required to restore the two least significant bits of x^r.

If the above formulas are calculated in that order, the s most significant bits of x^{r+1} and x^{r+2} can also be recovered at the same time according to (5.4).

The two least significant bits of all the secret values of x^{r+2} are unknown, which leaves r+1 as the only round for which all the secret values are completely recovered allowing us to recalculate all the values for all the following and preceding rounds.

It turns out that due to the perfectly cyclic structure of the VSC Type 3, there is no difference in which order the secret values are recovered on each round.

5.3. Missing bits

In cases where an even number is encountered among x_i such that more than s of its least significant bits is zeroes, some of the secret values cannot be fully recovered by the above process. Although such situations represent a minority of cases, they must be taken into account. If the amount of available plaintext-ciphertext pairs is limited, in the worst case it's

always possible to brute-force the few missing bits verifying the result rolling the cipher backward or forward for as many rounds as necessary.

6. Possible attack modifications

The algorithm presented in this paper is the simplest of the attacks differing from each other only slightly. The same algorithm can be applied to other single-round variants with minor modifications and insignificant differences in complexity. In practical applications where no more than two subsequent outputs are available or where only parts of outputs are available, the secret state can be recovered partially and the rest may be guessed and verified with enough available cipher outputs to validate the guesses. Most importantly, the last output block only provides the least significant two bits for each 32-bit word thus leaving only $2^{w/2}$ bits or less to guess if that output is not available or if it's available only partially. Guessing those bits would reduce the number of required subsequent output blocks by one. For example, for the 128-bit variant of the cipher it results in 8 bits of information which can be easily guessed and the guess verified given at least 8 more bits of output from anywhere else in the stream. In order to verify each guess the cipher can be trivially rolled forward or backward any number of rounds.

Replacing the calculation of the round r+1 values in the step 5.4 with a more sophisticated cipher roll-back thus beginning all the calculations with the round r values instead of r+1 would further reduce the number of required output blocks by one.

Thus the VSC Type 3 can be successfully broken with only two subsequent blocks of known plaintext and with at least $2^{w/2}$ bits of known plaintext available from anywhere else in the stream.

The VSC key recovery process described in this paper successfully recovers keys for all single-round variants of the VSC Type 3 (VSC-128, VSC-256, VSC-512 and VSC-1024). Our key recovery process can also be extended with statistical methods to break multiple-round variants; however, thanks to the trivial algebraic structure of the VSC it would seem more reasonable to exploit the simple algebraic relationships between individual bits in the cipher.

7. Conclusion

The high-speed reduced-round VSC variants are found insecure and should not be used for any cryptographic applications. Due to the simple algebraic nature of the cipher, its weak design and the amount of information released on every round, it is trivial to break and tweaks like VSC-128S or other minor alterations to the algorithm do not pose any threat to the attacker. All other reduced-round variants of the VSC will remain trivially breakable. The algorithm presented in this paper can also trivially break some of the multiple-round VSC variants (see Appendix B) and can assist in breaking the full 8 rounds with statistical, 'Mod n' or algebraic attacks.

We must admit that the fact that the single-round VSC is so trivially breakable would be obvious to any cryptographer skilled in the art. It is a great surprise to us that the cipher has reached such a broad distribution and was titled as secure without a more thorough review.

The VSC authors' claims of high speed of a secure cipher are exposed as false, especially taking into account the fact that performance of the VSC ciphers [unlike ciphers operating in counter mode] cannot be improved by pipelining. *The more secure 8-round VSC is of course 8 times slower than the hardware VSC speed advertised by ChaosWare.*

8. References

[1] 梅野健、金成主、長谷川晃朗、"VSC128 仕様書", March 2004.
http://www.chaosware.com/vsc128.pdf

[2] 梅野健、金成主、長谷川晃朗、"VSC128S 仕様書", September 2004.
http://www.chaosware.com/vsc128s.pdf

[3] J. Kelsey, B. Schneier, and D. Wagner, "Mod n Cryptanalysis, with Applications against RC5P and M6", Fast Software Encryption, Sixth International Workshop Proceedings (March 1999), Springer-Verlag, 1999, pp. 139-155.

Appndix A

In this appendix we include the original demonstration by Benjamin Gittins, Synaptic Laboratories, that the claimed 25 Gigabit per second hardware performance of the VSC-128 cipher can only be achieved by the single-round VSC variant.

The paper "Scalable Chaotic Cipher Chip and its Performance Evaluation for Hardware Implementation" (Authors names are published in Japanese) claims that VSC-128 implemented in Xilinx XCV21000-5 achieves 7.033 Gbps with output of 128 bits per "operation" at 57.05 MHz. If we accept the bandwidth claim, the clock speed must be more accurately presented as 54.9 MHz.
The clock speed of synchronous circuits is bound by the critical path through the circuit. The critical path of the VSC algorithm can be simplified to the equation, where X, C and Y are 32-bit values:

$$Y = (X * (2*X + ((C \text{ AND NOT } 3) \text{ OR } 1))) \text{ AND } (2^{32} - 1)$$

The above equation was implemented in Verilog with the inputs and outputs to the function registered. We synthesised the module using the Xilinx development environment. Static timing analysis was performed on the circuit after place-and-route.

The multiplication mod 2^{32} circuitry was mapped to three 18x18 dedicated multipliers executing in parallel in combination with additional carry-logic circuitry. The register-to-register delay was measured to be ~13.2 ηs translating to the clock speed of ~75.8 MHz. The figure below represents bounds on performance that can be achieved without the use of pipelining by a single round.

Operation	Clock Speed
18x18 multiplication	105 MHz
Addition 16-bit	239 MHz
Addition 64-bit	114 MHz

Official Xilinx XCV21000-5 performance figures
(Virtex-II Platform FPGA Design Sheet)

The Xilinx XV2V1000 distributes 40 multipliers across 4 columns with 10 instances in each column. The difference of the simplified circuit and the published results of about 21 MHz can be heavily attributed to the additional wire-latencies incurred by the distribution of the twelve 18x18 multipliers across 2 or 3 columns and the transposition operation.

Thus it seems reasonable that only a single-round variant can be implemented at the advertised clock speed within the advertised area.

To perform 8 rounds of VSC at the advertised clock speed would require an 8-stage pipeline, placing a register between each round. Eight rounds of VSC would require 96 instances of the 18x18 multipliers. Unfortunately it is not possible to map the 96 multiplier resources required for the 8-round variant pipelined to the 40 multipliers available on the XV2V1000! Although such implementation would be able to encrypt a large amount of data at the advertised speed over multiple channels, the speed of each individual channel would still be limited to $1/8^{th}$ of the advertised speed.

Our conclusion is that it is impossible for the advertised VSC hardware implementation to perform 8 rounds at the advertised high speed of 25 Gigabit per second and that it is the single-round VSC variant advertised as achieving the high encryption speed.

Appendix B

Remarks regarding the unique differences of the VSC Type 1 key recovery:

- the *s*-bit rotation operation performed *after* the multiplication layer results in the loss of the top *s* bits of the $x_{w/2}$ word; those *s* bits must be either brute-forced or recovered with an additional available output; the same process must be repeated on every step while recovering all the subsequent secret 32-bit blocks;

- the *s*-bit rotation operation performed *after* the multiplication layer also causes the least significant bits shift by *s* bits to the left thus allowing for incremental accumulation of the amount of information available on the last step of the recovery process until x_0 is completely recovered for one of the states, followed by a direct calculation of the remaining blocks;

- the *s*-bit rotation operation performed *after* the multiplication layer also makes the loss of the most significant bits slow down by *s* bits on every step when encountering even numbers, with no loss of information when encountering 32-bit blocks with up to *s* least significant zeroes;

- due to the fact that this variant releases a half of its internal state consisting of 32-bit blocks that affect each other sequentially, multiple-round variants of this type can be trivially broken for $r < w/4$ rounds;

Remarks regarding the unique differences of the VSC Type 2 key recovery:

- the *s*-bit rotation operation performed *before* the multiplication layer results in a complete recovery of the entire secret register on every step unless the number of least significant zeroes is greater than *s*, in which case the lost most significant bits may not be instantly recovered like in the VSC Type 1 and the number of lost bits grows rapidly with each step;

- the *s*-bit rotation operation performed *before* the multiplication layer also results in a somewhat slower recovery of x_0 on the last step, followed by a direct calculation of the remaining blocks;

- due to the fact that this variant releases a half of its internal state consisting of 32-bit blocks that affect each other sequentially, multiple-round variants of this type can be trivially broken for $r < w/4$ rounds.

Note: the VSC-128S has been released only a few days prior to the original publication of this paper and therefore no details of the key recovery process for this variant can be included, however the preliminary review shows that only insignificant changes to the VSC Type 3 key recovery process are required.

With minor modifications this algorithm can break VSC ciphers Type 1 and 2 trivially up to $w/4$ rounds, thus providing trivial key recovery for instance for a 7-round VSC-1024, easily extendable further to break the “full” 8 rounds. The security margin of 8 rounds seems to be too small for the larger sizes of VSC claiming higher security.