HS1-SIV (Draft v2)

Submitted and designed by
Ted Krovetz[1]
ted@krovetz.net
12 May 2014

HS1-SIV uses a new PRF called HS1 to provide authenticated encryption via Rogaway and Shrimpton's SIV mode [6]. HS1 (mnemonic for "Hash-Stream 1") is designed for high software speed on systems with good 32-bit processing, including Intel SSE, ARM Neon, and 32-bit architectures without SIMD support. HS1 uses a universal hash function to consume arbitrary strings and a stream cipher to produce its output.

This document defines HS1 and how to use it in an SIV construction. HS1 takes an arbitrary input string and IV and produces a pseudorandom string of any desired length. Each different (input, IV) pair supplied to HS1 yields an independent pseudorandom stream with high probability. SIV, as defined in [6], uses a block-cipher-based PRF to create a synthetic IV (an SIV) from given associated data and plaintext. The SIV is then used to encrypt the plaintext using a block-cipher-based encryption scheme. HS1-SIV instead uses HS1 to instantiate SIV mode. Ignoring many details for the moment, if A is the associated data, M is the plaintext, and N is HS1's IV, then the SIV is defined as the first 16 bytes of HS1($A||M$, N) and ciphertext C is defined as all but the first 16 bytes of HS1(SIV, N) xor'ed with M. The SIV and ciphertext are bundled together to create the final ciphertext. If (A, M, N) is repeated, then an observer knows this fact because the scheme is deterministic and the SIV will be identical, but no security degradation otherwise occurs. Supplying N as a nonce thus improves security by masking repeated encryptions.

HS1 does its work by pairing an almost-universal hash function with a stream cipher. When given an (input, IV) pair, HS1 uses the hash function to hash the input, it then xor's this hash result with the stream cipher's key and uses the HS1 IV as the stream cipher's IV. The stream cipher produces as many bytes as desired. As long as a (hash result, IV) pair is never repeated, and the stream cipher is secure against related-key attacks, the stream cipher will produce independent pseudorandom output streams. We introduce a new hash HS1-Hash which we use for the almost-universal phase of HS1, and Bernstein's Chacha is used as the stream cipher [1].

1 Specification

The figures in this document fully specify HS1-SIV with the exception of the Chacha stream cipher. We use the Chacha version specified in *ChaCha20 and Poly1305 for IETF protocols* [5], and attach it to make this document self-contained. The interface defined in that document has Chacha take four inputs: A 32-byte key, an initial counter value, a 12-byte IV, and a plaintext. The ciphertext produced is the same length as the plaintext and is pseudorandom. This document adopts that interface. $\mathsf{Chacha}[r](K, c, N, X)$ indicates the r-round Chacha encryption of X using key K, initial counter value c, and IV N. When we simply want an n-byte pseudorandom string, we let X be n zero bytes.

[1] Department of Computer Science, California State University, Sacramento CA 95819. This work supported by the generous support of NSF grants CNS-1314592 and CNS-0904380.

```
$\mathsf{HS1\text{-}SIV\text{-}Encrypt}[b,t,r,\ell](K,M,A,N)$
Inputs:
        $K$, a non-empty string of up to 32 bytes
        $M$, a string shorter than $2^{64}$ bytes
        $A$, a string shorter than $2^{64}$ bytes
        $N$, a 12-byte string
Output:
        $(T,C)$, strings of $\ell$ and $|M|$ bytes, respectively
Algorithm:
1.      $\mathbf{k} = \mathsf{HS1\text{-}subkeygen}[b,t,r,\ell](K)$
2.      $M' = \mathsf{pad}(16,A) \,||\, \mathsf{pad}(16,M) \,||\, \mathsf{toStr}(8,|A|) \,||\, \mathsf{toStr}(8,|M|)$
3.      $T = \mathsf{HS1}[b,t,r](\mathbf{k},M',N,\ell)$
4.      $C = M \oplus \mathsf{HS1}[b,t,r](\mathbf{k},T,N,64+|M|)[64,|M|]$
```

Figure 1: Encryption. *The ℓ-byte string T serves as authenticator for A and M, and serves as IV for the encryption of M. If N is a nonce, then repeat encryptions yield different T and C. Algorithm parameters b,t,r, and ℓ effect security and performance.*

1.1 Parameters

There are four parameters used throughout this specification, b,t,r,ℓ. Parameter b specifies the number of bytes used in part of the hashing algorithm (larger b tends to produce higher throughput on longer messages). Parameter t selects the collision level of the hashing algorithm (higher t produces higher security and lower throughput). Parameter r specifies the number of internal rounds used by the stream cipher (higher r produces higher security and lower throughput). Parameter ℓ specifies the byte-length of synthetic IV used (higher ℓ improves security and increases ciphertext lengths by ℓ bytes). The following table names parameter sets.

Name	b	t	r	ℓ
`hs1-siv-lo`	64	2	8	8
`hs1-siv`	64	4	12	16
`hs1-siv-hi`	64	6	20	32

1.2 Notation

The algorithms in this document manipulate integers, vectors of integers, and strings of bytes. Vector and string indices begin with zero. If $\mathbf{v}$ is a vector, then $\mathbf{v}[a]$ is its a-th element and $\mathbf{v}[a,n]$ is the n-element subvector beginning at index a: $(\mathbf{v}[a], \mathbf{v}[a+1], \ldots, \mathbf{v}[a+n-1])$. Similarly, if S is a string, then $S[a,n]$ is the n-byte substring of S beginning at index a. The number of elements in $\mathbf{v}$ and bytes in S is indicated $|\mathbf{v}|$ and $|S|$. Concatenation is accomplished using "$||$" (eg, $(1,2,3)||(4,5) = (1,2,3,4,5)$ and `0xf0`$||$`0xa8` = `0xf0a8`). The bitwise exclusive-or of same-length strings A and B is $A \oplus B$. A string of k zero-bytes is represented 0^k. $\mathsf{pad}(n,S) = S||0^k$ where k is the smallest non-negative integer making the length of $S||0^k$ a non-negative multiple of n bytes. $\mathsf{toStr}(n,x)$ is the n-byte unsigned little-endian binary representation of integer x (eg, $\mathsf{toStr}(2,3) =$ `0x0300`). $\mathsf{toInts}(n,S)$ is the vector of integers obtained by breaking S into n-byte chunks and little-endian interpreting each chunk as an unsigned integer (eg, $\mathsf{toInts}(2,$`0x05000600`$) = (5,6)$).

```
$\mathsf{HS1}[b,t,r](\mathbf{k},M,N,y)$
Inputs:
    $\mathbf{k}$, a vector $(K_S, \mathbf{k}_N, \mathbf{k}_P, \mathbf{k}_A)$, where
        $K_S$, is a string of 32 bytes,
        $\mathbf{k}_N$, is a vector of $b/4 + 4(t-1)$ integers from $\mathbb{Z}_{2^{32}}$,
        $\mathbf{k}_P$, is a vector of $t$ integers from $\mathbb{Z}_{2^{60}}$, and
        $\mathbf{k}_A$, is a vector of $3t$ integers from $\mathbb{Z}_{2^{64}}$
    $M$, a string of any length
    $N$, a 12-byte string
    $y$, an integer in $\mathbb{Z}_{2^{38}}$
Output:
    $Y$, a string of $y$ bytes
Algorithm:
1.  $A_i = \mathsf{HS1\text{-}Hash}[b,t](\mathbf{k}_N[4i, b/4], \mathbf{k}_P[i], \mathbf{k}_A[3i,3], M)$ for each $0 \le i < t$
2.  $Y = \mathsf{Chacha}[r](\mathsf{pad}(32, A_0 \mid\mid A_1 \mid\mid \ldots \mid\mid A_{t-1}) \oplus K_s), 0, N, 0^y)$
```

Figure 2: HS1 PRF. *Hash M a total of t times with different keys and combine the result with the stream cipher's key.*

Given vectors of integers $\mathbf{v}_1$ and $\mathbf{v}_2$, $\mathsf{NH}(\mathbf{v}_1, \mathbf{v}_2) = \sum_{i=1}^{n/4}((\mathbf{v}_1[4i-3] + \mathbf{v}_2[4i-3]) \times (\mathbf{v}_1[4i-1] + \mathbf{v}_2[4i-1]) + (\mathbf{v}_1[4i-2] + \mathbf{v}_2[4i-2]) \times (\mathbf{v}_1[4i] + \mathbf{v}_2[4i]))$ where $n = \min(|\mathbf{v}_1|, |\mathbf{v}_2|)$ and is always a multiple of four.

2 Security Goals

HS1-SIV provides confidentiality of plaintexts and integrity of ciphertexts, associated data and public message numbers. No private message numbers are employed by HS1-SIV. If we say an adversary can win an attack against HS1-SIV by (i) guessing the key in use or (ii) causing two different encryptions to use the same synthetic IV, then the following table summarizes an adversary's probability of success over n encryptions, each of no more than 2^{32} bytes, or n key guesses.

Name	Key Search	SIV Collision
`hs1-siv-lo`	$n/2^{256}$	$n^2/2^{56} + n^2/2^{64}$
`hs1-siv`	$n/2^{256}$	$n^2/2^{112} + n^2/2^{128}$
`hs1-siv-hi`	$n/2^{256}$	$n^2/2^{168} + n^2/2^{256}$

SIV is designed to provide the maximum possible robustness against nonce reuse. HS1-SIV maintains full integrity and confidentiality, except for leaking collisions of (plaintext, associated data, public message number) via collisions of ciphertexts. If two different (plaintext, associated data, public message number) triples produce the same SIV, then forgeries become possible.

```
HS1-Hash$[b,t](\mathbf{k}_N, k_P, \mathbf{k}_A, M)$
Inputs:
        $\mathbf{k}_N$, is a vector of $b/4$ integers from $\mathbb{Z}_{2^{32}}$,
        $k_P$, is an integer from $\mathbb{Z}_{2^{60}}$
        $\mathbf{k}_A$, is a vector of 3 integers from $\mathbb{Z}_{2^{64}}$ (Not used when $t \le 4$)
        $M$, a string of any length
Output:
        $Y$, an 8 byte (if $t \le 4$) or 4 byte (if $t > 4$) string
Algorithm:
1.      $n = \max(\lceil |M|/b \rceil, 1)$
2.      Let $M_1, M_2, \ldots, M_n$ be strings so that $M_1 || M_2 || \cdots || M_n = M$ and
            $|M_i| = b$ for each $1 \le i < n$.
3.      $\mathbf{m}_i = \mathsf{toInts}(4, \mathsf{pad}(16, M_i))$ for each $1 \le i \le n$
4.      $a_i = \mathsf{NH}(\mathbf{k}_N, \mathbf{m}_i) \ \mathbf{mod}\ 2^{60} + (|M_i| \ \mathbf{mod}\ 16)$ for each $1 \le i \le n$
5.      $h = k_P^n + a_1 k_P^{n-1} + a_2 k_P^{n-2} + \ldots + a_n k_P^0 \ \mathbf{mod}\ (2^{61} - 1)$
6.      if $(t \le 4)$ $Y = \mathsf{toStr}(8, h)$
7.      else $Y = \mathsf{toStr}(4, (\mathbf{k}_A[0] + \mathbf{k}_A[1] \times (h \ \mathbf{mod}\ 2^{32}) + \mathbf{k}_A[2] \times (h \ \mathbf{div}\ 2^{32}))\ \mathbf{div}\ 2^{32})$
```

Figure 3: *The hash family HS1-Hash is $(1/2^{28} + \ell/b2^{60})$-AU for all M up to ℓ bytes, when $\mathbf{k}_N$ and k_P are chosen randomly and $t \le 4$. The hash family is $(1/2^{28} + 1/2^{32} + \ell/b2^{60})$-SU when additionally $\mathbf{k}_A$ is randomly chosen and $t > 4$.*

3 Security Analysis

HS1 is a composition of HS1-Hash, an almost-universal hash function, with Chacha, a stream cipher. HS1-Hash is itself a composition of two hashes. Similar to techniques used in UMAC and VMAC [3, 4], the NH hash is used to reduce the input by a fixed ratio to an intermediate string which is then hashed to a fixed size by a polynomial evaluation. It uses well-known and well-studied techniques. The inner-product hashes inputs of length bm bytes to ones of length $8m$ bytes with a collision probability of no more than 2^{-28}. The resulting $8m$-byte string is hashed using a standard polynomial hash modulo the prime $2^{61} - 1$. Because the polynomial's key is chosen over a restricted set (for performance reasons), the final collision probability is no more than $m2^{-60} + 2^{-28}$. To reduce the chance of collision, this hashing procedure is repeated t times, with different keys, for an ultimate collision probability of no more than $(m2^{-60} + 2^{-28})^t$.

We conjecture Chacha is secure against related-key attacks: an adversary with reasonably limited resources would have difficulty distinguishing between $f(X, N) = \mathsf{Chacha}(X \oplus K, 0, N, 0^\ell)$ and a randomly chosen function with the same signature $\{0,1\}^{256} \times \{0,1\}^{96} \to \{0,1\}^\ell$ when K is a high-entropy 256-bit string unknown to the adversary and ℓ is any positive number. This conjecture is supported by numerous statements by Bernstein and the way Chacha and Salsa "cores" are deployed in Chacha, Salsa and Rumba. Bernstein's statements have been directed toward Salsa, Chacha's predecessor, but Chacha is simply Salsa with a couple of small improvements, and it is widely believed that Chacha inherits all of Salsa's positive security features. Bernstein's statements and uses are consistent with the belief that the Salsa and Chacha cores are simply pseudorandom functions mapping 48-byte inputs to 64-byte outputs. If this is true, then the conjecture that Chacha is secure against related-key attacks is immediate. In *Salsa20 security* Bernstein writes: "The reader might guess that Salsa is highly resistant to related-key attacks."

```
HS1-Subkeygen$[b, t, r, \ell](K)$
Inputs:
    $K$, a non-empty string of up to 32 bytes
Output:
    $(K_S, \mathbf{k}_N, \mathbf{k}_P, \mathbf{k}_A)$, where
        $K_S$, is a 32-byte string,
        $\mathbf{k}_N$, is a vector of $b/4 + 4(t-1)$ integers from $\mathbb{Z}_{2^{32}}$,
        $\mathbf{k}_P$, is a vector of $t$ integers from $\mathbb{Z}_{2^{60}}$, and
        $\mathbf{k}_A$, is a vector of $3t$ integers from $\mathbb{Z}_{2^{64}}$
Algorithm:
1.  $ChachaLen = 32$
2.  $NHLen = b + 16(t-1)$
3.  $PolyLen = 8t$
4.  $ASULen = 24t$
5.  $y = ChachaLen + NHLen + PolyLen + ASULen$
6.  $K' = (K||K||K||K||\ldots)[0, 32]$
7.  $N = \mathsf{toStr}(12, b2^{48} + t2^{40} + r2^{32} + \ell 2^{16} + |K|)$
8.  $T = \mathsf{Chacha}[r](K', 0, N, 0^y)$
9.  $K_S = T[0, ChachaLen]$
10. $\mathbf{k}_N = \mathsf{toInts}(4, T[ChachaLen, NHLen])$
11. $\mathbf{k}_P = \mathsf{map}(\mathbf{mod}\ 2^{60}, \mathsf{toInts}(8, T[ChachaLen + NHLen, PolyLen]))$
12. $\mathbf{k}_A = \mathsf{toInts}(8, T[ChachaLen + NHLen + PolyLen, ASULen])$
```

Figure 4: *HS1-Subkeygen takes any length key and uses Chacha to produce all internal keys needed by HS1.*

In *Response to "On the Salsa core function"* Bernstein claims that the core is designed "to eliminate all visible structure". In the Rumba compression function, adversaries are allowed to provide any selected 48-byte inputs to the cores, and in Salsa and Chacha the cores are used simply in counter mode to produce their pseudorandom streams.

HS1 is essentially defined $\text{HS1}(K, h, X, N, \ell) = \mathsf{Chacha}(h(X) \oplus K, 0, N, 0^\ell)$ where K is a high-entropy key and h is from HS1-Hash. As long as $(h(X) \oplus K, N)$ values are distinct, Chacha will always produce independent pseudorandom streams. HS1 is designed to make it unlikely that $(h(X) \oplus K, N)$ pairs ever repeat. A user supplied nonce is used for N with each message, and even if this facility is misused by giving the same nonce repeatedly, $h(X)$ will be distinct with high probability over a sequence of distinct X values.

4 Features

HS1-SIV is designed to have the following features.

Competitive speed on multiple architectures. HS1-SIV is designed to exploit 32-bit multiplication and SIMD processing, which are well-supported on almost all current CPUs. This ensures a consistent performance profile over a wide range of processors, including modern embedded ones.

```
HS1-SIV-Decrypt$[b, t, r, \ell](K, (T, C), A, N)$
Inputs:
    $K$, a non-empty string of up to 32 bytes
    $(T, C)$, an $\ell$-byte string and a string shorter than $2^{64}$ bytes, respectively
    $A$, a string shorter than $2^{64}$ bytes
    $N$, a 12-byte string
Output:
    $M$, a $|C|$-byte string, or AuthenticationError
Algorithm:
1.  $\mathbf{k} = \mathsf{HS1\text{-}subkeygen}[b, t, r, \ell](K)$
4.  $M = C \oplus \mathsf{HS1}[b, t, r](\mathbf{k}, T, N, 64 + |C|)[64, |C|]$
2.  $M' = \mathsf{pad}(16, A) \mid\mid \mathsf{pad}(16, M) \mid\mid \mathsf{toStr}(8, |A|) \mid\mid \mathsf{toStr}(8, |M|)$
3.  $T' = \mathsf{HS1}[b, t, r](\mathbf{k}, M', N, \ell)$
4.  if $(T = T')$ then return $M$
4.  else return AuthenticationError
```

Figure 5: Decryption. *The ℓ-byte string T serves as authenticator for A and M, and serves as IV for the decryption of C. Algorithm parameters b, t, r, and ℓ effect security and performance.*

Provable security. HS1-Hash and SIV are based on well-known and proven constructions [4, 6]. The only security assumption needed is that the Chacha stream cipher is a good pseudorandom generator and secure against related-key attacks.

Nonce misuse resistant. No harm is done when a nonce is reused, except that it is revealed whether corresponding (associated data, plaintext) pairs have been repeated. If (associated data, plaintext) pairs are known to never repeat, no nonce need be used at all.

Scalable. Different security levels are available for different tasks, with varying throughput.

General-purpose PRF. The general nature of HS1 makes it useful for a variety of tasks, such as entropy harvesting, random generation, and other IV-based encryption and authentication schemes. A single software library could provide multiple services under a single key by simply partitioning the nonce space and providing access to HS1.

With the exception of provable security, all of the above features are improvements over GCM.

5 Design Rationale

HS1-SIV is designed to exploit a variable-input-length/variable-output-length PRF. This abstraction allows the PRF and its application to be designed separately. Although the PRF could be used in several different ways to realize authenticated encryption, SIV is a natural fit because its definition requires variable input-lengths to be consumed, variable output lengths to be produced, and the intermediate value–the synthetic IV–to be exposed. Furthermore, SIV enjoys provable security, allowing the security focus of HS1-SIV to be entirely on HS1.

HS1 is conceived as a fast almost-universal hash function composed with a fast stream cipher, with little integration overhead. HS1-Hash is heavily influence by VMAC [4], which targets beefy CPUs, but with the assumption that 32-bit multiplication–with or without SIMD acceleration–will be the maximum-size available efficient multiplication. This allows good performance on a wide range of CPU's; from modern embedded CPU's (the 43 thousand gate ARM Cortex-M3 has 32-bit multiplication with a 64-bit result) to modern Intel processors (Intel's current AVX instruction set can process four 32-bit multiplies in a single instruction). The result is a hash that performs nearly as well on 32-bit CPUs as 64-bit ones. Bernstein's Chacha is chosen as the stream cipher for three reasons: it is one of the fastest stream ciphers in software on our target systems, its key setup is simple and fast, and it is secure against related-key attacks.

The designer has not hidden any weaknesses in this cipher.

6 Intellectual Property

The submitter knows of no known patents, patent applications, planned patent applications, or other intellectual-property constraints relevant to use of HS1-SIV. If any of this information changes, the submitter/submitters will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

7 Consent

The submitter/submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitter/submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitter/submitters understand that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitter/submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitter/submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitter/submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

8 Revision History

The submitted v1 version mistakenly defined Prod. This version replaces it with NH and updates a few sentences to accommodate the change. Also, v1 mistakenly gave bitlengths in the table of Section 1.1, and they are now listed as bytelengths.

References

[1] Daniel Bernstein. ChaCha, a variant of Salsa20. Workshop Record of SASC 2008: The State of the Art of Stream Ciphers. ECRYPT, 2008. Also available at `http://cr.yp.to/chacha.html`.

[2] Daniel Bernstein. The Salsa20 family of stream ciphers. In: New stream cipher designs: the eSTREAM finalists. Springer 2008. Also available at `http://cr.yp.to/snuffle.html`

[3] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz and Phillip Rogaway. UMAC: Fast and secure message authentication. Advances in Cryptology (CRYPTO 1999), Springer, 1999. Also available at `http://krovetz.net/csus`.

[4] Ted Krovetz. Message authentication on 64-bit architectures. Selected Areas of Cryptography (SAC 2006), Springer, 2007. Also available at `http://krovetz.net/csus`.

[5] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF protocols. IETF Internet Draft. `http://datatracker.ietf.org/doc/draft-nir-cfrg-chacha20-poly1305`. Accessed 15 March 2014.

[6] Phillip Rogaway and Tom Shrimpton. Deterministic Authenticated-Encryption: A Provable-Security Treatment of the Keywrap Problem. EUROCRYPT 2006. Springer, 2006. Also available at `http://www.cs.ucdavis.edu/~rogaway/papers`.