Cryptanalysis of $22\frac{1}{2}$ rounds of Gimli

Mike Hamburg*

August 1, 2017

Abstract

Bernstein et al. have proposed a new permutation, Gimli, which aims to provide simple and performant implementations on a wide variety of platforms. One of the tricks used to make Gimli performant is that it processes data mostly in 96-bit *columns*, only occasionally swapping 32-bit words between them.

Here we show that this trick is dangerous by presenting a distinguisher for reduced-round Gimli. Our distinguisher takes the form of an attack on a simple and practical PRF that should be nearly 192-bit secure. Gimli has 24 rounds. Against $15\frac{1}{2}$ of those rounds, our distinguisher uses two known plaintexts, takes about 2^{64} time and uses enough memory for a set with 2^{64} elements. Against $19\frac{1}{2}$ rounds, the same attack uses three non-adaptively chosen plaintexts, and uses twice as much memory and about 2^{128} time. Against $22\frac{1}{2}$ rounds, it requires about $2^{138.5}$ work, 2^{129} bits of memory and $2^{10.5}$ non-adaptively chosen plaintexts. The same attack would apply to $23\frac{1}{2}$ rounds if Gimli had more rounds.

Our attack does not use the structure of the SP-box at all, other than that it is invertible, so there may be room for improvement. On the bright side, our toy PRF puts keys and data in different positions than a typical sponge mode would do, so the attack might not work against sponge constructions.

1 Introduction

Permutation-based cryptography is attractive for lightweight devices because it can use a single cryptographic primitive for many applications. However, many cryptographic permutations perform well only on lightweight devices, or only on large devices, or only in hardware.

Bernstein et al. recently proposed Gimli in order to bridge this divide [1]. Gimli performs passably well in hardware, in lightweight devices and in large devices, making it an attractive option for future deployments.

One of Gimli's innovations is that it processes most of its data in 96-bit columns (consisting of three 32-bit words), and only swaps one word from each column every other round. This improves performance on tiny devices, which can process one column at a time with little loss

*Rambus Security Division

in performance. But we will show that it is dangerous, and leads to attacks on the reduced-round permutation in practical applications. Our attack is not merely a distinguisher, but a key recovery attack against a realistic deployment of Gimli. Furthermore, it only uses two chosen plaintexts, and does not use the structure of the SP-box at all.

2 The Gimli permutation

Gimli is fully specified in [1]. Here we need only its overall structure. Let W be the space of all 32-bit words, and let $C := W^3$ be the space of all 96-bit columns. The SP-box is a permutation $P : C \to C$. For any permutation $Q : S \to S$, let

$$Q \times Q : S^2 \to S^2$$

be the permutation defined by

$$(Q \times Q)(x, y) := (Q(x), Q(y))$$

Let the permutation SmallSwap : $C^2 \to C^2$ be defined as

$$\begin{aligned} \text{SmallSwap} \quad & ((a, b, c), (d, e, f)) \\ := \quad & ((d, b, c), (a, e, f)) \end{aligned}$$

Likewise, let BigSwap : $C^4 \to C^4$ be defined as

$$\begin{aligned} \text{BigSwap} \quad & ((a, b, c), (d, e, f), (g, h, i), (j, k, l)) \\ := \quad & ((g, b, c), (j, e, f), (a, h, i), (d, k, l)) \end{aligned}$$

Let $\text{Swap}_i : (W^3)^4 \to (W^3)^4$ be SmallSwap $\times$ SmallSwap (plus a round constant which doesn't affect our analysis) if $i \equiv 0 \pmod 4$, or BigSwap if $i \equiv 2 \pmod 4$, or the identity otherwise. Then the ith Gimli round is a permutation $G_i : C^4 \to C^4$ is

$$G_i := \text{Swap}_i \circ (P \times P \times P \times P)$$

An important observation for our attack is that 4 consecutive rounds of Gimli process the two halves of the state separately until the final BigSwap. Let $H_4 : C^2 \to C^2$ be defined as

$$H_4 := (P \times P) \circ (P \times P) \circ \big[\text{SmallSwap} \circ (P \times P)\big] \circ (P \times P)$$

so that four rounds of Gimli can be written as

$$R_4 := G_{4i+2} \circ G_{4i+3} \circ G_{4i+4} \circ G_{4i+5} = \text{BigSwap} \circ (H_4 \times H_4)$$

Gimli's rounds count from 24 down to 1. Here we will analyze a reduced-round variant which performs rounds 21 down to 2, less the final swap, so that it breaks down naturally into blocks of R_4. Call this function $\text{Gimli}_{19.5}$. If we align to the beginning or end of Gimli, this attack breaks one fewer round but otherwise works the same way. Our attack uses only H_4 and H_4^{-1}, which we treat as black boxes, and BigSwap. We ignore the internals of H_4, P and SmallSwap.

3 A PRF based on Gimli

We will analyze a natural pseudorandom function based on Gimli. This function takes a 192-bit key $k \in C^2$ and a 192-bit input $x \in C^2$, and produces a 192-bit output $y \in C^2$ by

$$F(k, x) := \text{snd}(\text{Gimli}(k, x))$$

where snd denotes the second half of the state. This function should have nearly 192 bits of security against generic attacks[1]. However, we will show that with $\text{Gimli}_{19.5}$, it only has 128 bits of security, and for $\text{Gimli}_{22.5}$ it has only 138.5 bits of security. The reason is that there is very little communication between the first and second halves of the state.

Note that Gimli's word order goes along rows first, rather than down columns first. Furthermore, most sponge modes xor keys into data before calling the permutation. So our attack is unlikely to apply as written to, e.g., libhydrogen.

3.1 The 12-round core

For our pseudorandom F, the first four rounds and last 3 rounds of Gimli do almost nothing. The first 4 rounds are

$$R_4 = \text{BigSwap} \circ (H_4 \times H_4)$$

and the last 3.5 rounds are $H_4 \times H_4$. That is,

$$F_{19.5}(k, x) = H_4(\text{snd}(R_4^3(\text{BigSwap}(H_4(k), H_4(x)))))$$

Since H_4 is efficiently invertible, the security of $F_{19.5}$ is equivalent to that of

$$\text{Core}_{12}(k, x) := \text{snd}(R_4^3(\text{BigSwap}(k, x)))$$

4 A meet-in-the-middle attack

Here is our meet-in-the-middle attack on Core_{12}, which therefore also breaks $F_{19.5}$. A helpful diagram of the attack on $F_{19.5}$ is shown in Figure 1.

For this attack, we need only to query $\text{Core}_{12}(k, x)$ for two values x_1, x_2 which have the same top row. That is, let

$$x_i := (a, b_i, c_i), (d, e_i, f_i)$$

where $a, b_{\{1,2\}}, c_{\{1,2\}}, d, e_{\{1,2\}}$ and $f_{\{1,2\}}$ are arbitrary.[2] Now, let the functions $K_{i,j} : C^3 \to C^3$ be defined as

$$K_{i,j}((a, b, c), (d, e, f)) := (k_{i,j,1}, b, c), (k_{i,j,2}, e, f)$$

[1] We didn't prove this. The best generic attack we found was to query about $2^{192}/6$ plaintexts, at which point a balls-in-bins analysis suggests that there should be about a 6-way collision in ciphertexts. Then we can get the key by guessing values for $\text{fst}(\text{Gimli}(k, x))$. This takes about about $2^{192}/6$ guesses in expectation, for a total of $2^{192}/3$ work.

[2] Thus this attack applies to Core_{12} in counter mode, but for $F_{19.5}$ it requires chosen plaintexts.

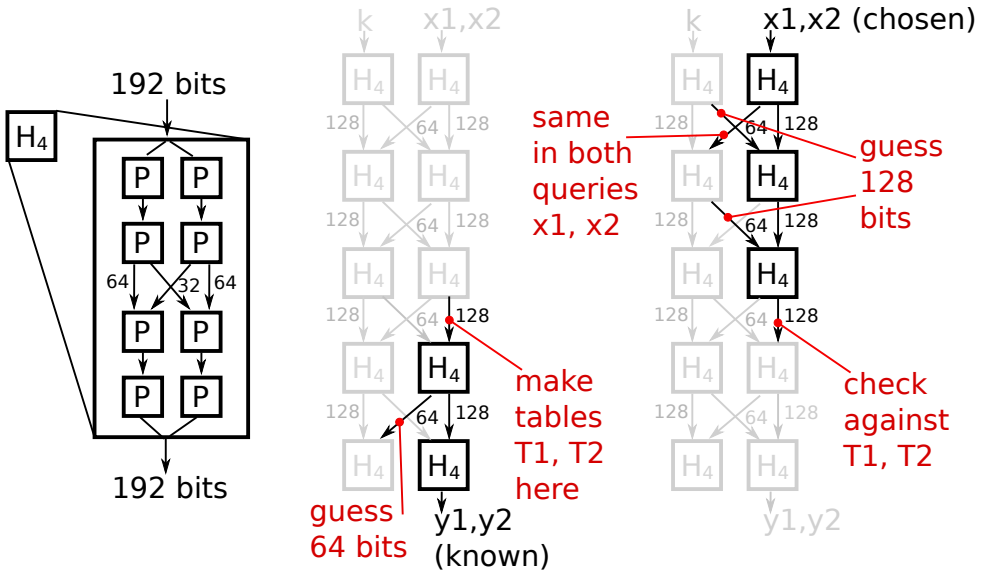

Figure 1: Meet-in-the-middle attack against GIMLI: 2^{128} complexity for 19.5 rounds.

where $k_{i,j,1}$ and $k_{i,j,2}$ are the ith pair of unknown 32-bit words which come from the key half in the jth chosen plaintext block. We likewise define their duals

$$\bar{K}_{i,j}((a,b,c),(d,e,f)) := (\bar{k}_{i,j,1},b,c),(\bar{k}_{i,j,2},e,f)$$

where $\bar{k}_{i,j,1}$ and $\bar{k}_{i,j,2}$ were the words swapped out for $k_{i,j,1}$ and $k_{i,j,2}$. We will use $\bar{K}_{i,j}$ when running GIMLI rounds backwards. Note that $K_{1,1} = K_{1,2}$ always and $K_{2,1} = K_{2,2}$ because of how we chose the plaintexts; abbreviate these as K_1 and K_2 respectively. We see that

$$\begin{aligned} y_1 := \text{Core}_{12}(k,x_1) &= (K_{4,1} \circ H_4 \circ K_{3,1} \circ H_4 \circ K_2 \circ H_4 \circ K_1)(x_1) \\ y_2 := \text{Core}_{12}(k,x_2) &= (K_{4,2} \circ H_4 \circ K_{3,2} \circ H_4 \circ K_2 \circ H_4 \circ K_1)(x_2) \end{aligned}$$

To reverse this function, we will guess the the two words swapped out at the end of each of these applications. Let

$$\text{bottom}((a,b,c),(d,e,f)) := (b,c,e,f)$$

be the bottom 4 words of a given two columns. We make two tables T_1, T_2 containing sets of 2^{64} 128-bit values each, where

$$T_j := \{\ \text{bottom}(H_4^{-1}(\bar{K}_{4,j}(y_j)))\ \}$$

for all 2^{64} possible value pairs that could have been injected by $\bar{K}_{4,j}$. We then see that

$$\text{bottom}(H_4(K_2(H_4(K_1(x_j))))) \quad \in \quad T_j$$

for $j \in \{1,2\}$. We can then exhaust over the 2^{128} values which could have been injected by K_1 and K_2, and check if the results are in T_1 and T_2. Since the probability of a random element

being in both tables is 2^{-128}, we expect to see only a few matching values. With a third chosen plaintext, we can reduce that to one matching value with high probability.

After this entire procedure, we will have a single value (or perhaps a few candidates) for the top 64 bits of the key after the first H_4 step. Then we can brute-force the remaining unknown 128 bits, and output the key that it is consistent with $F_{19.5}(x_1)$ and $F_{19.5}(y_1)$.

4.1 138.5-bit attack on 22.5 (or 23.5) rounds

The same attack breaks Core_{16} and thus 23.5-round Gimli with $2^{138.5}$ work and 2^{129} bits of memory. We use $2^{10.5}$ chosen plaintexts x_i, as before all with the same top two words. This gives us $2^{10.5}$ samples y_i, where

$$y_j := \text{Core}_{16}(k, x_j) \quad = \quad (K_{5,j} \circ H_4 \circ K_{4,j} \circ H_4 \circ K_{3,j} \circ H_4 \circ K_2 \circ H_4 \circ K_1)(x_j)$$

We again build sets T_j of 128-bit values from y_j, but now guessing both $\bar{K}_{5,j}$ and $\bar{K}_{4,j}$. There would be 2^{128} elements in each set, but because the function to build the set is not a permutation, there will only about $2^{128} \cdot (1 - 1/e)$ distinct elements. Each set takes 2^{128} bits of memory to describe, for a total of $2^{138.5}$ bits of memory and as much work. Running on the queries one at a time reduces the memory usage to 2^{129} bits at a small cost in compute time: 2^{128} for the current table, and 2^{128} to determine which keys have been ruled out.

After building the sets, we again guess the words injected by K_1 and K_2, and compute

$$(H_4 \circ K_2 \circ H_4 \circ K_1)(x_j)$$

for each $i \in [0, 2^{10.5}]$. For each such j, with probability about $1/e$ this value won't be in T_j, so we can reject the guess. Our

$$\log_{1-1/e} 2^{-128} \approx 2^{10.5}$$

queries will be enough to reject all but the correct guess, and perhaps a few wrong ones. We can then brute-force the remaining 128 bits of the key as before.

This attack doesn't apply to 23.5 rounds of the real Gimli: it breaks what would be rounds 25 through 2.5, but Gimli has only 24 rounds. So it only breaks 22.5 rounds of the real Gimli.

4.2 64-bit attack on 15.5 rounds

The same attack breaks 15.5 rounds with about 2^{64} work. We make the same table, but now we only have to guess the 64 bits injected by K_1. This is only a 64-bit distinguisher and not a key recovery attack. Probably the key can be recovered with a little additional work using the structure of H_4, but we didn't investigate this.

5 Implementation

To confirm that our attacks works, implemented it on Gimli variants with reduced word sizes. We implemented the 15.5-round attack on a version which uses 16-bit words, taking about 2^{32} time and $2^{32} \cdot 64$ memory instead of 2^{64} time and $2^{64} \cdot 128$ memory. We ran the attack on a Skylake NUC we had lying around. We had to tweak the attack to take more time and less

memory by using approximate sets (i.e. Bloom filters), because our NUC didn't have $2^{32} \cdot 64$ bits = 32 GiB of memory, but otherwise it worked.

We implemented the 23.5 round attack on a Gimli variant with 8-bit words, taking about $2^{37.5}$ time and 2^{33} bits of memory (1 GiB) instead of about $2^{138.5}$ time and 2^{129} bits of memory.

For both attacks, we just confirmed that they gave the right K_1, and didn't bother brute-forcing the rest of the key. Both attacks completed within a few hours on one core.

6 Future work

It would be interesting to turn the 64-bit attack into a key recovery attack. Also, we are curious whether properties of the SP-box can be used to improve this attack, or if it can be changed from a table-based attack to a rho attack. Finally, it would be interesting to see whether the same attacks can be used to break other uses of reduced-round Gimli, e.g. finding collisions in the hash function, and whether the attack can be adapted to sponge modes.

7 Conclusion

Gimli's slow diffusion is a serious weakness. The linear layer should be replaced, and preferably should be performed every round instead of every other round. This will reduce Gimli's performance, but it is important for security. In the mean time, Gimli is not appropriate for scenarios requiring more than 128-bit security. Fortunately, our attack is mitigated by the modes used in a typical sponge construction.

8 Acknowledgements

Special thanks to Mark Marson for editing this paper.

References

[1] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli: a cross-platform permutation. Accepted to CHES, 2017. http://eprint.iacr.org/2017/630.