Bit attacks

D. J. Bernstein
University of Illinois at Chicago
Running CubeHash8/1 with 64 bit output over 2 different datasets give me the same hash under Visual Studio. Using the code from simple.c and call it the following way:
memcpy(data, "AAAAAAAAAABBBB\0\0\0\0\0", 16);
Hash(64, data, 16, hash);
for (i = 0; i < 8; i++)
    printf("%02x", 0xff & hash[i]);
printf("\n");

memcpy(data, "AAAAAAAAAACBBB\0\0\0\0\0", 16);
Hash(64, data, 16, hash);
for (i = 0; i < 8; i++)
    printf("%02x", 0xff & hash[i]);
printf("\n");
As you can see, there is a minor difference in the dataset (first "B" replaced with a "C"). Running it produces:

379ec80069d7a71b
379ec80069d7a71b

Is this the winner of the final CubeHash prize?
Let’s look at what happened.

Programmer wants to hash a string $s$ with $n$ bytes.

Classic MD5 API:
“input has inputlen bytes.”
Okay: input = $s$;
     inputlen = $n$
Let's look at what happened.

Programmer wants to hash a string \( s \) with \( n \) bytes.

Classic MD5 API:
“input has inputlen bytes.”
Okay: \( \text{input} = s; \)
\[ \text{inputlen} = n \]

NIST SHA-3 API:
“data has databitlen bits.”
Okay: \( \text{data} = s; \)
\[ \text{databitlen} = 8 \times n \]
e.g. `databitlen = 128`

to hash 16 bytes:

<table>
<thead>
<tr>
<th>AAAA</th>
<th>AAAA</th>
<th>BBBB</th>
<th>0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAA</td>
<td>AAAA</td>
<td>CBBB</td>
<td>0000</td>
</tr>
</tbody>
</table>
e.g. `databitlen = 128` to hash 16 bytes:

```
AAAAAAAABBBB0000
AAAAAAACBBB0000
```

What if the programmer forgets to multiply by 8?

`databitlen = 16`:

```
AA  AAAAAAABBBB0000
AA  AAAAAAACBBB0000
```
Responding to my own message here. Found the bug and it was my mistake. I call Hash with the number of bytes for datalength, instead of the number of bits.
What fraction of programmers will forget to multiply by 8? Let’s say fraction is $1/F$.

Surely SHA-3 will be used in $> 1000$ network protocols.

Expect $> 1000/F$ cases of server programmer forgetting to multiply by 8.

Will this bug be caught by interoperability tests?
Standardizing a protocol requires an independent client implementation.

Still expect $> 1000/F^2$ cases of client programmer and independent server programmer forgetting to multiply by 8.
Standardizing a protocol requires an independent client implementation.

Still expect $> 1000/F^2$ cases of client programmer and independent server programmer forgetting to multiply by 8.

Typical tests will be passed. Protocol will be deployable. Last 7/8th of message will be trivially modifiable.

Security disaster!