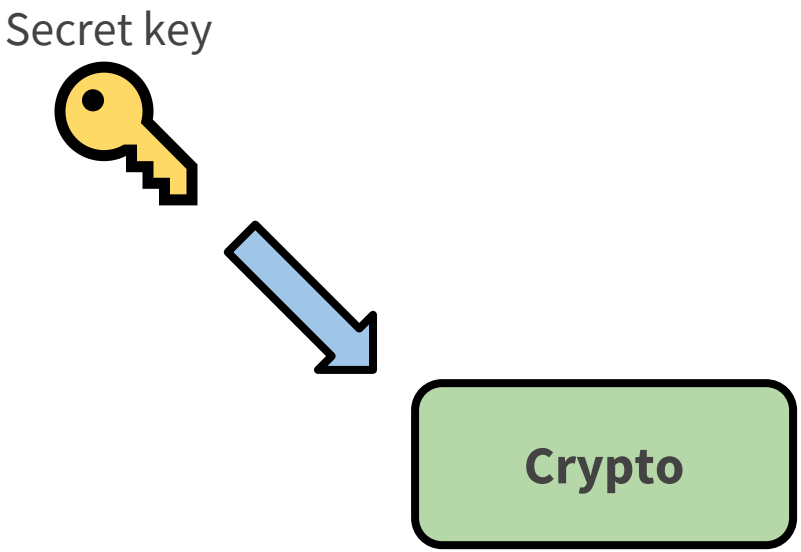
Constant-time programing in FaCT

Sunjay Cauligi, UC San Diego

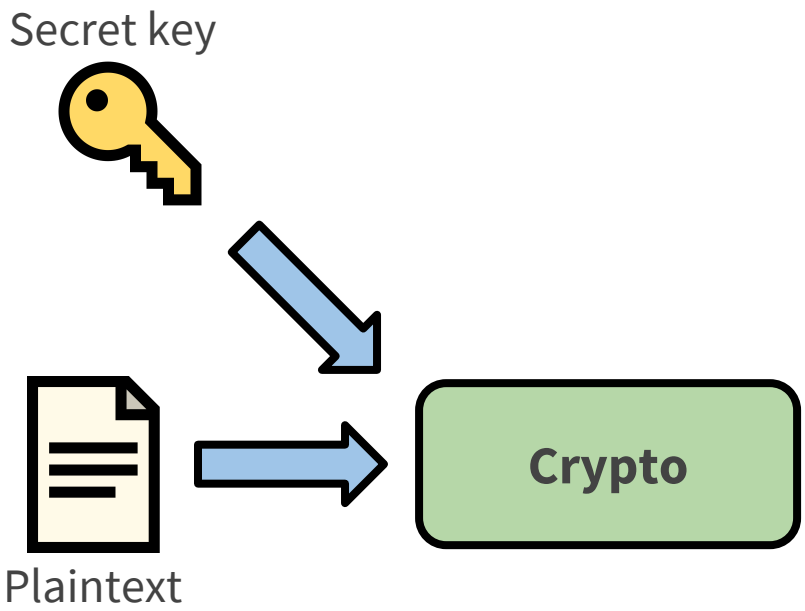
Fraser Brown, Ranjit Jhala, Brian Johannesmeyer, John Renner,
Gary Soeller, Deian Stefan, Riad Wahby, Conrad Watt

Timing side channels

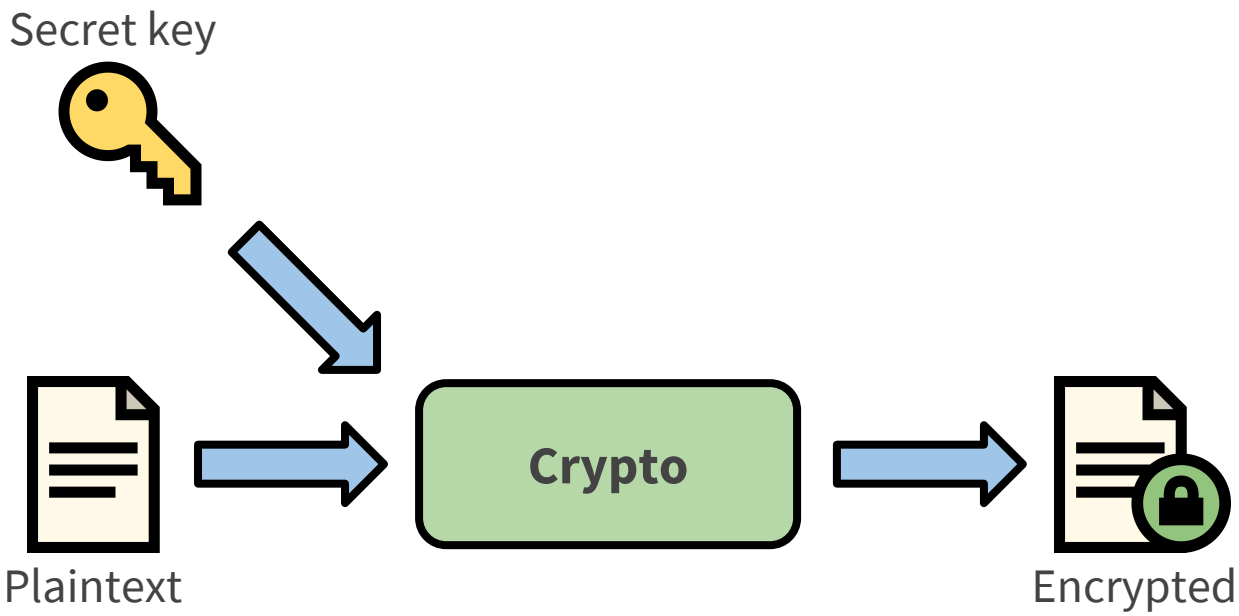
Timing side channels



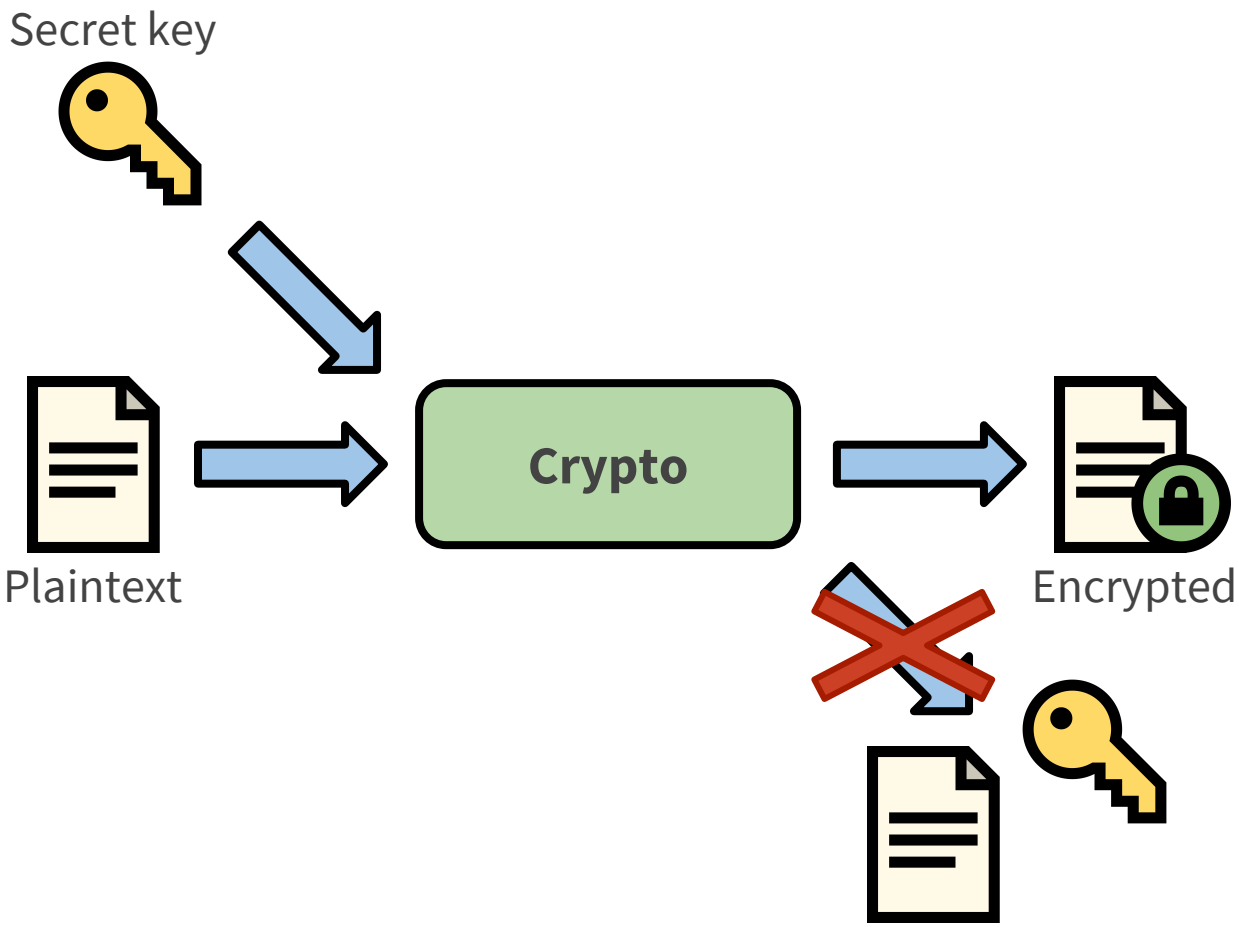
Timing side channels



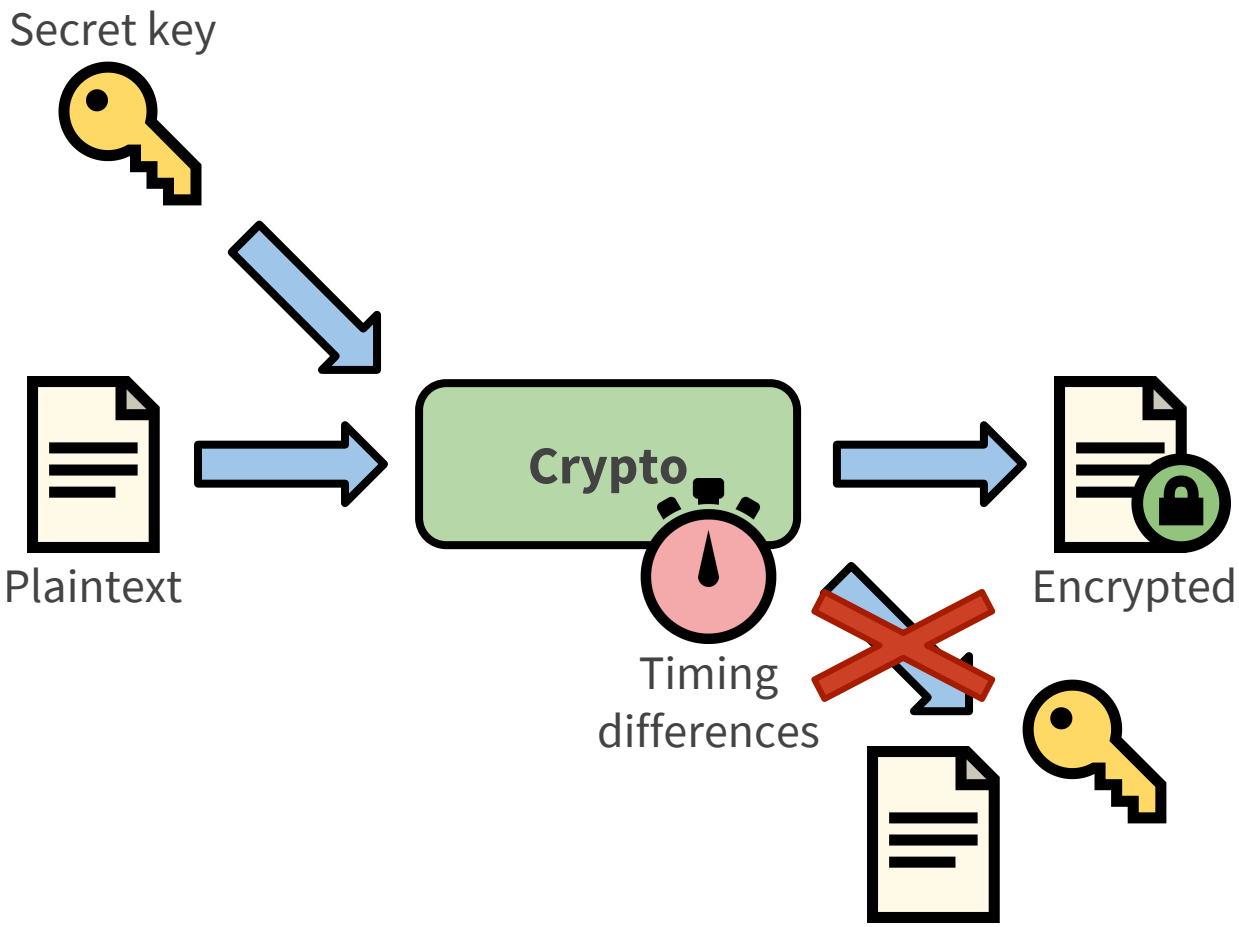
Timing side channels



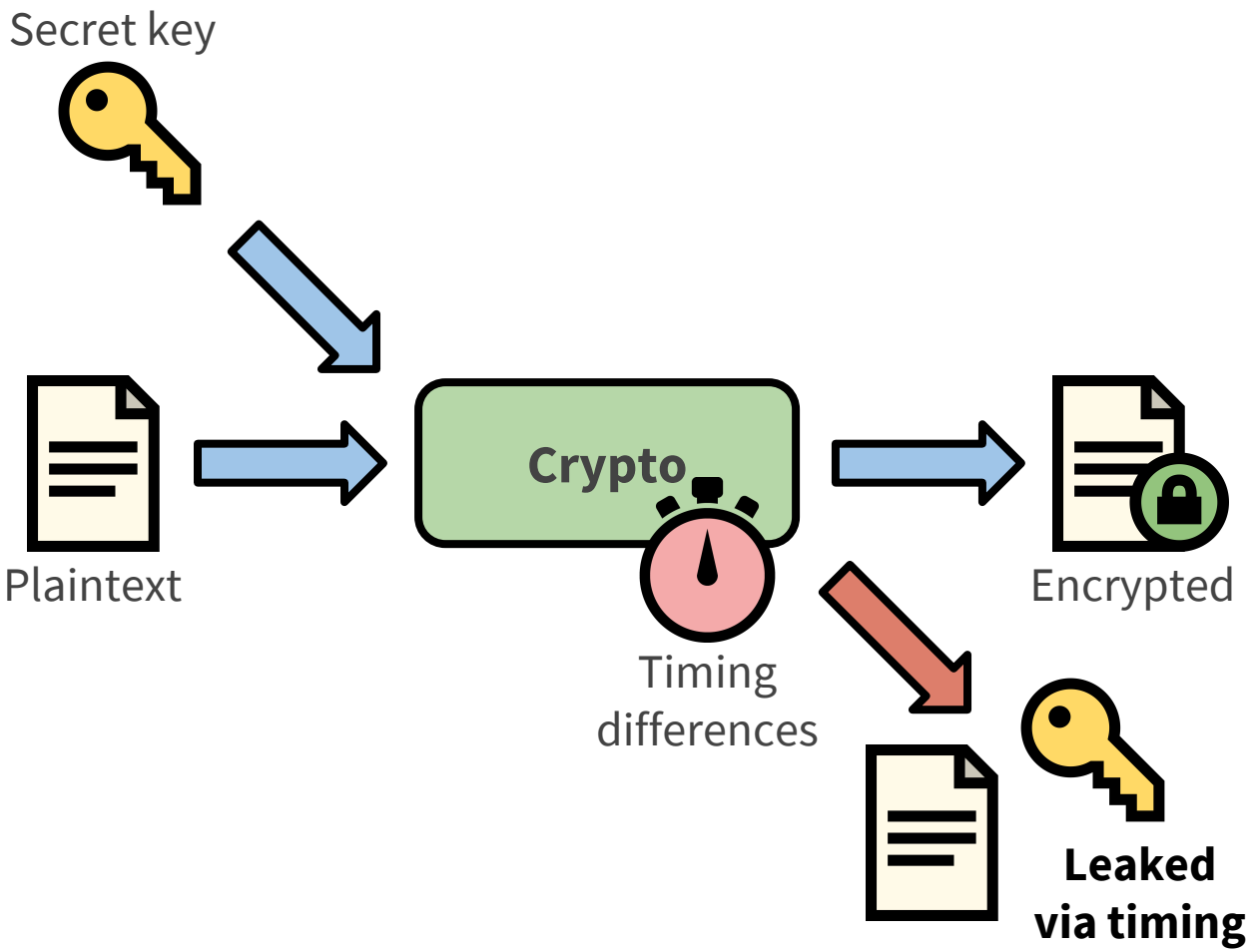
Timing side channels



Timing side channels



Timing side channels

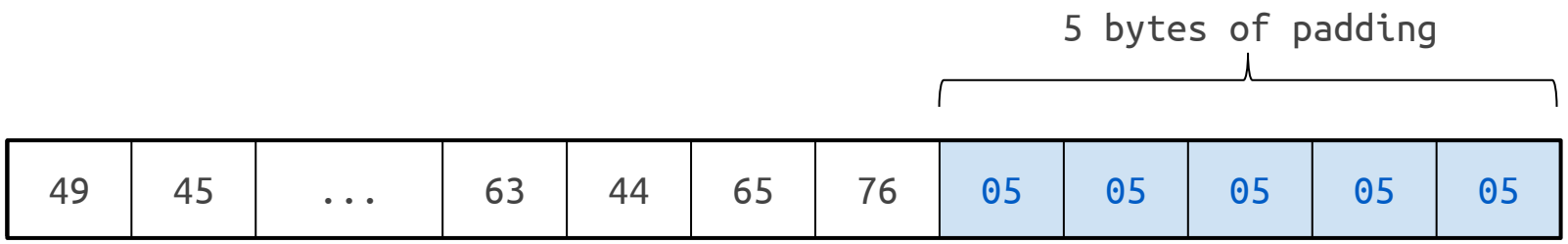


Constant-time coding example

- Check for valid padding
  - PKCS #7 padding
  - Each padding byte holds length of padding

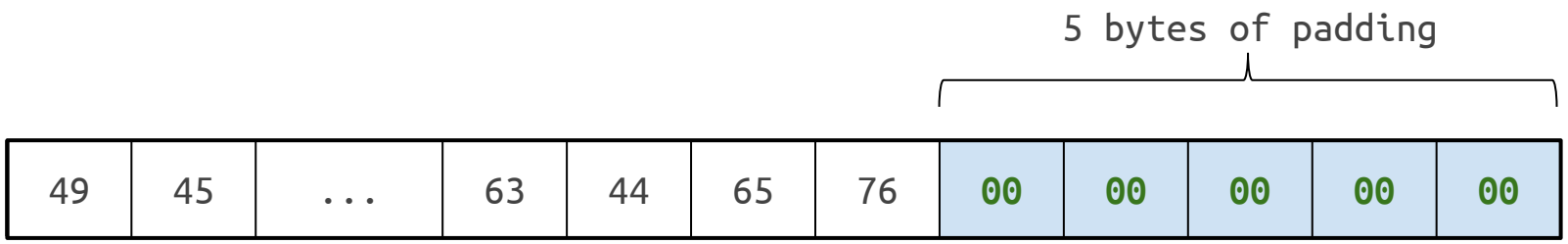
Constant-time coding example

- Check for valid padding
  - PKCS #7 padding
  - Each padding byte holds length of padding



Constant-time coding example

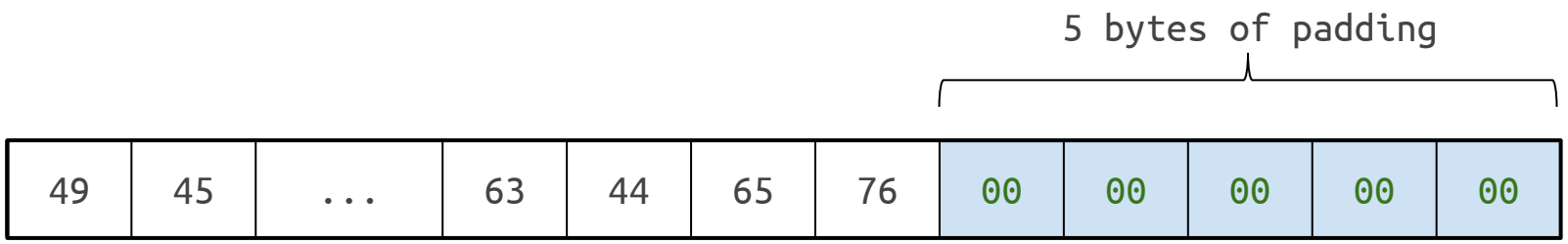
- Check for valid padding
  - PKCS #7 padding
  - Each padding byte holds length of padding
- Replace padding with null bytes
- Return padding length, or error



Constant-time coding example

- Check for valid padding
  - PKCS #7 padding
  - Each padding byte holds length of padding
- Replace padding with null bytes
- Return padding length, or error

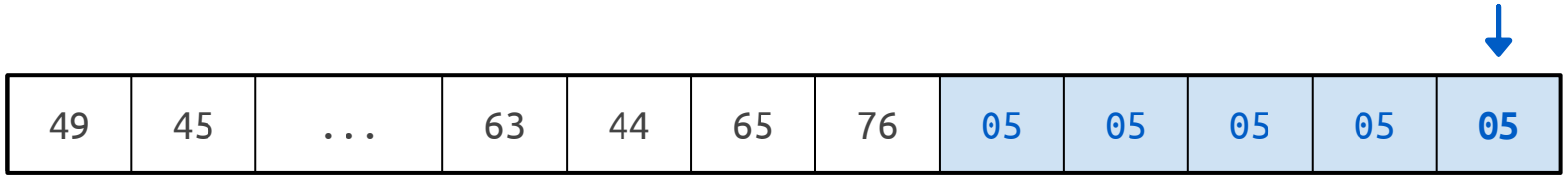
- Must be careful: buffer contents are secret
  - That includes padding!



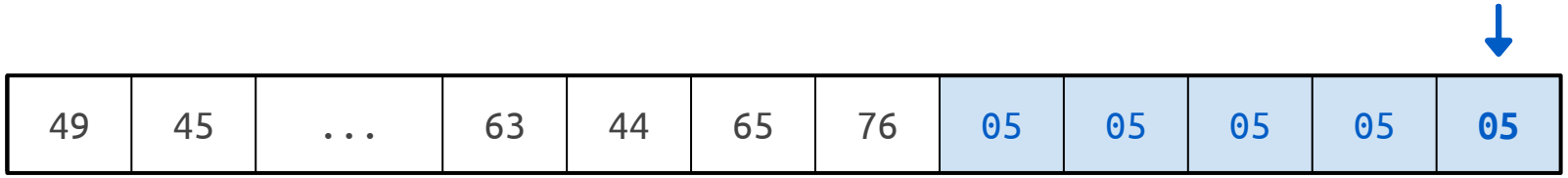
```
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```

49	45	...	63	44	65	76	05	05	05	05	05

```c
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```



```
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```



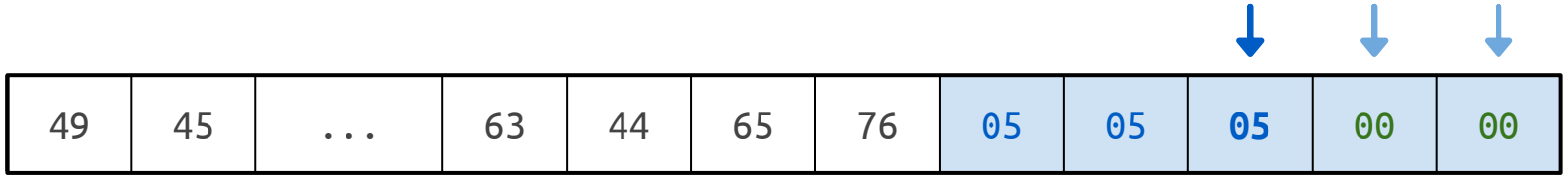
```
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```

49	45	...	63	44	65	76	05	05	05	05	**05**

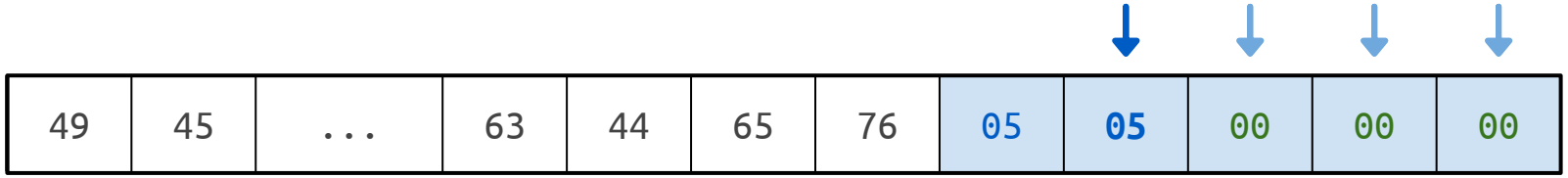
```
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```

49	45	...	63	44	65	76	05	05	05	05	00

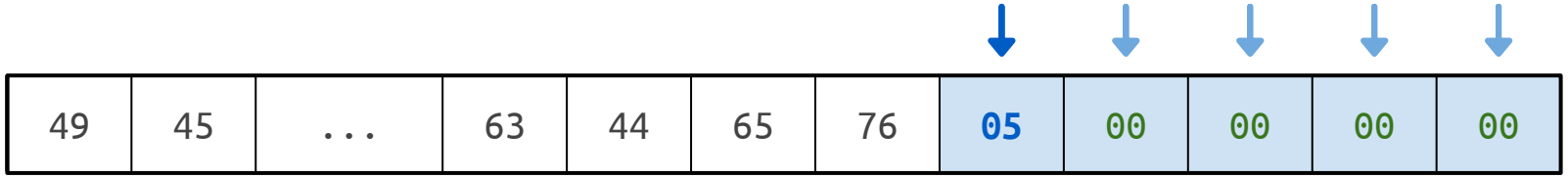
```
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```



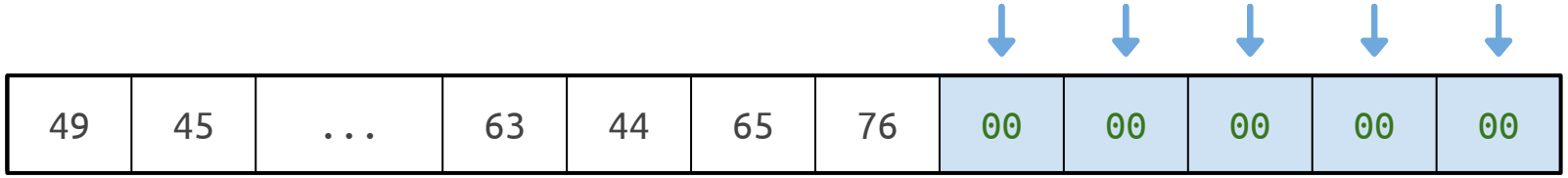
```c
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```



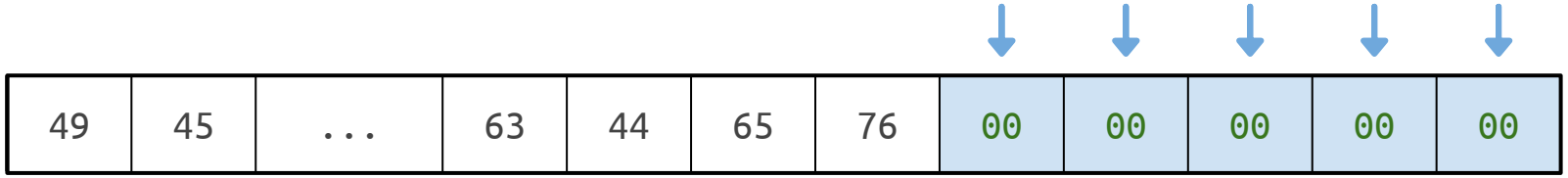
```c
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```



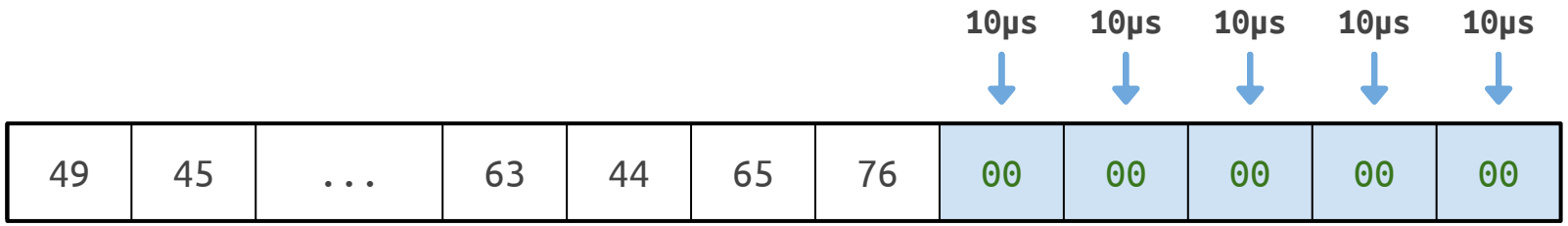
```
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```



```c
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```



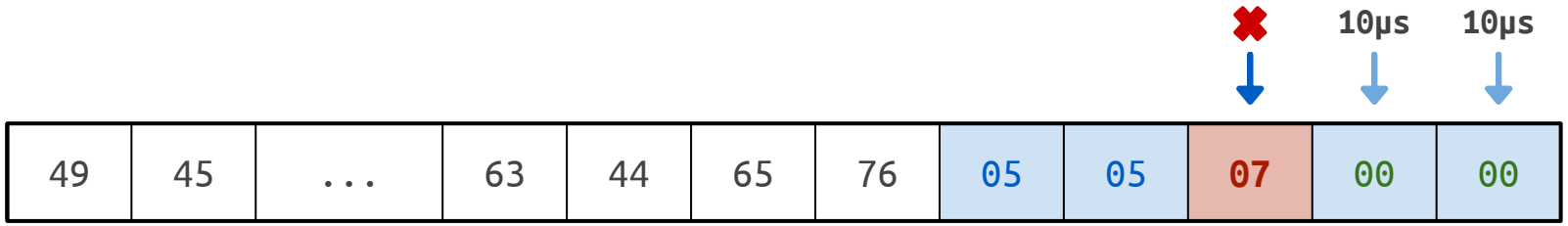
```c
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```



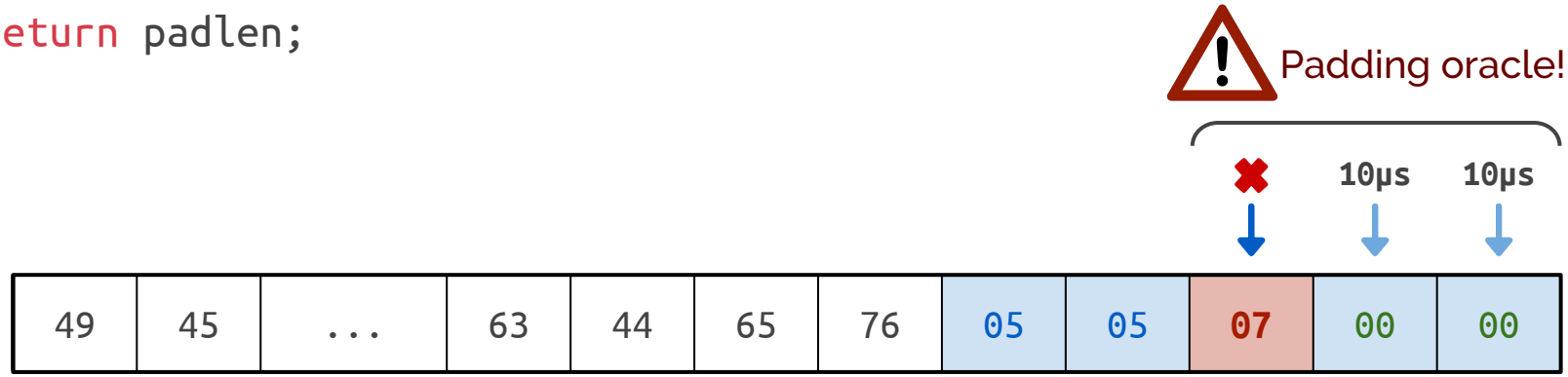
```
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```

49	45	...	63	44	65	76	05	05	**07**	05	05

```
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```

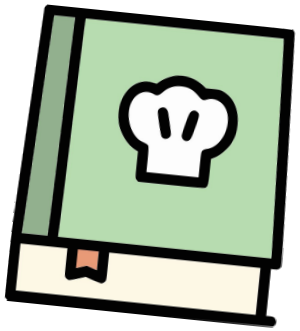


```
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```

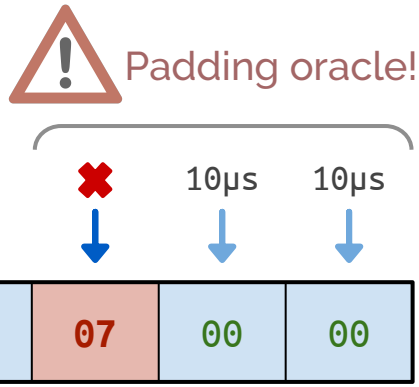


```
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```

It's dangerous to return early!

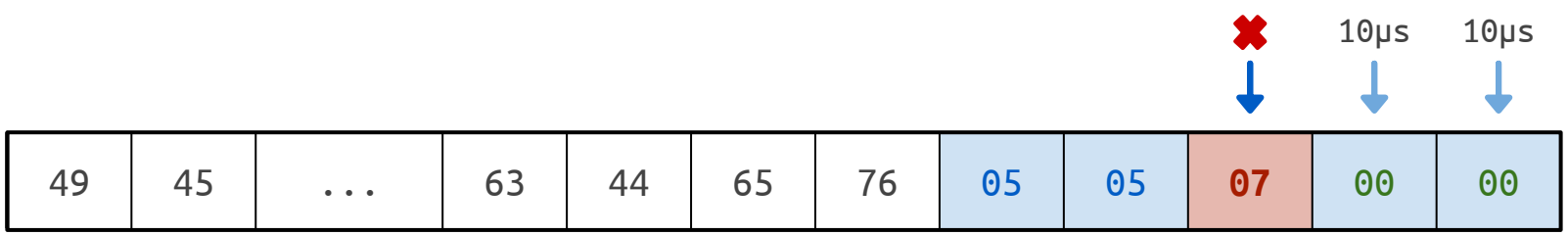


Use this instead.



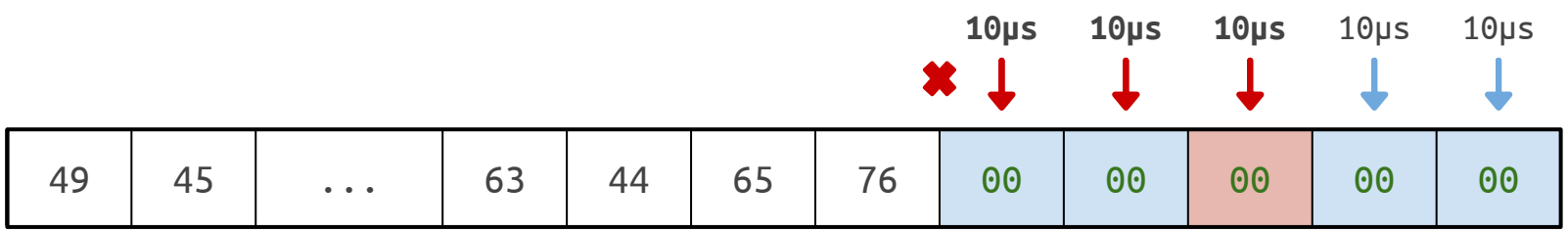
```c
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```

```c
int32_t remove_padding2(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      ok = 0;
    buf[buflen-i-1] = 0;
  }
  return ok ? padlen : -1;
}
```



```c
int32_t remove_padding(
    uint8_t* buf,
    uint32_t buflen) {

  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      return -1;
    buf[buflen-i-1] = 0;
  }
  return padlen;
}
```

```c
int32_t remove_padding2(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      ok = 0;
    buf[buflen-i-1] = 0;
  }
  return ok ? padlen : -1;
}
```



```c
int32_t remove_padding2(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      ok = 0;
    buf[buflen-i-1] = 0;
  }
  return ok ? padlen : -1;
}
```

49	45	...	63	44	65	76	05	05	05	05	05

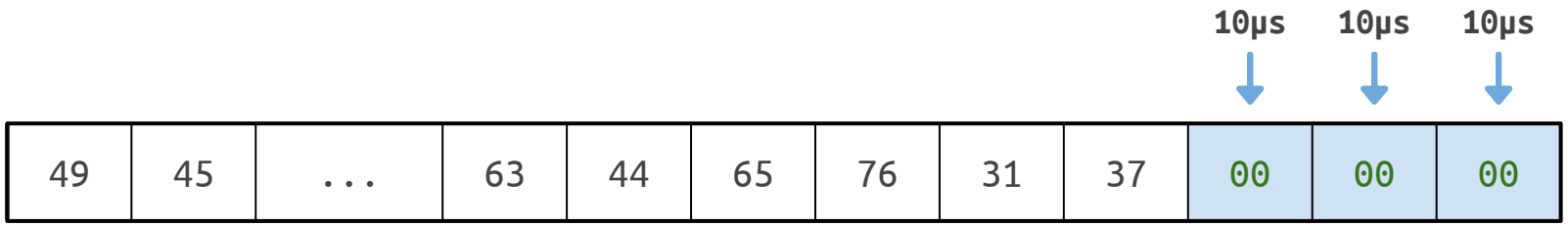
```
int32_t remove_padding2(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      ok = 0;
    buf[buflen-i-1] = 0;
  }
  return ok ? padlen : -1;
}
```

49	45	...	63	44	65	76	05	05	05	05	05

```c
int32_t remove_padding2(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      ok = 0;
    buf[buflen-i-1] = 0;
  }
  return ok ? padlen : -1;
}
```

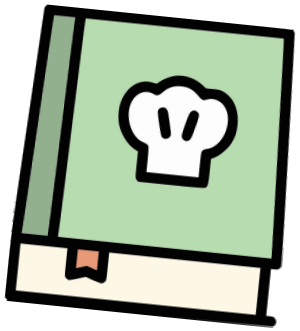
49	45	...	63	44	65	76	31	37	**03**	**03**	**03**

```c
int32_t remove_padding2(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      ok = 0;
    buf[buflen-i-1] = 0;
  }
  return ok ? padlen : -1;
}
```

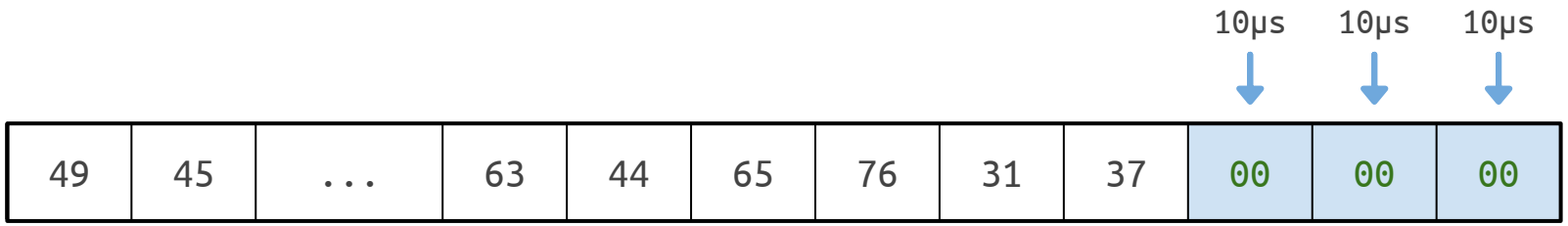


```
int32_t remove_padding2(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      ok = 0;
    buf[buflen-i-1] = 0;
  }
  return ok ? padlen : -1;
}
```

It's dangerous to bound loops with secrets!

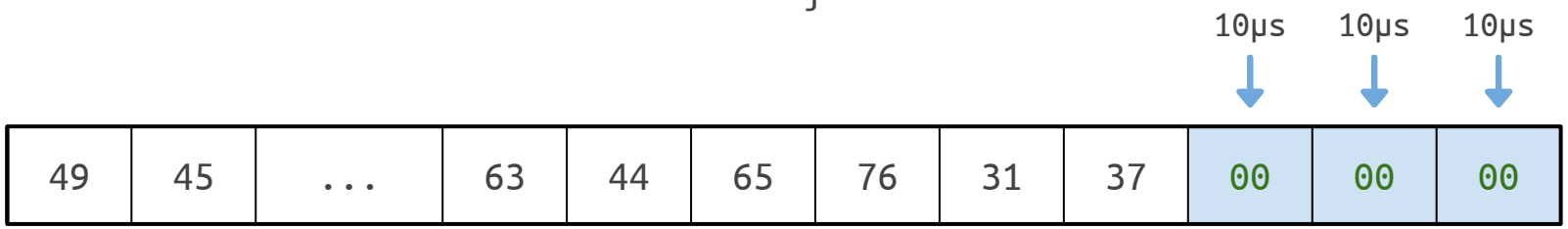


Use this instead.



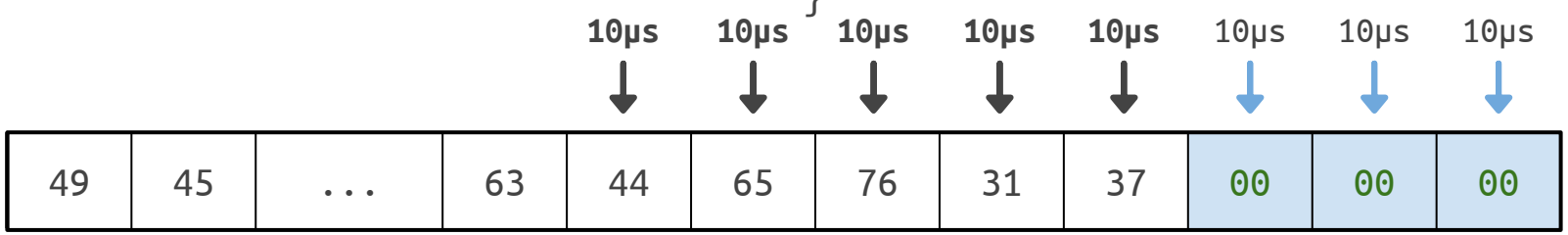
```
int32_t remove_padding2(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      ok = 0;
    buf[buflen-i-1] = 0;
  }
  return ok ? padlen : -1;
}
```

```
int32_t remove_padding3(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    if (i >= buflen - padlen) {
      if (b != padlen)
        ok = 0;
      b = 0;
    }
    buf[i] = b;
  }
  return ok ? padlen : -1;
}
```

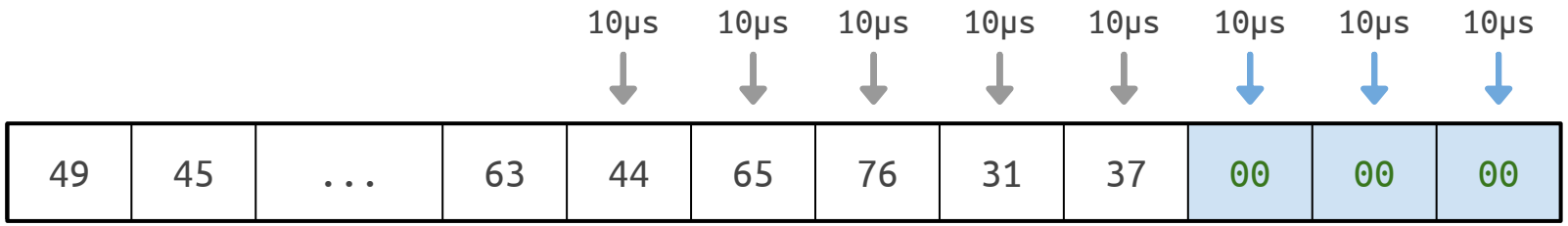


```c
int32_t remove_padding2(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = 0; i < padlen; i++) {
    if (buf[buflen-i-1] != padlen)
      ok = 0;
    buf[buflen-i-1] = 0;
  }
  return ok ? padlen : -1;
}
```

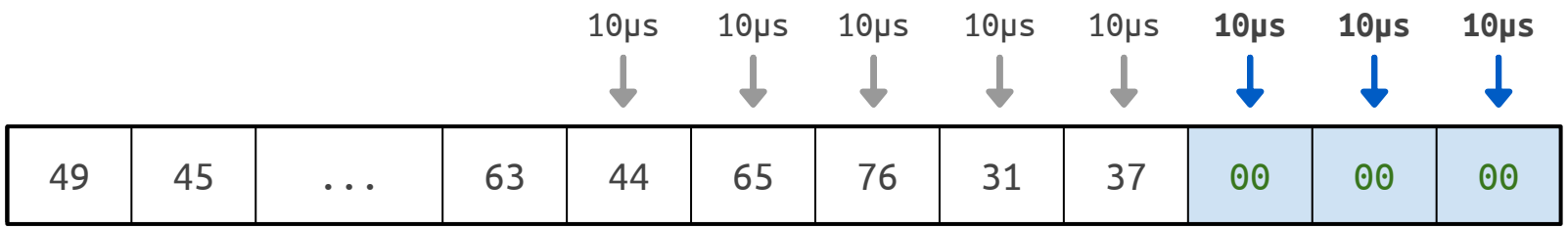
```c
int32_t remove_padding3(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    if (i >= buflen - padlen) {
      if (b != padlen)
        ok = 0;
      b = 0;
    }
    buf[i] = b;
  }
  return ok ? padlen : -1;
}
```



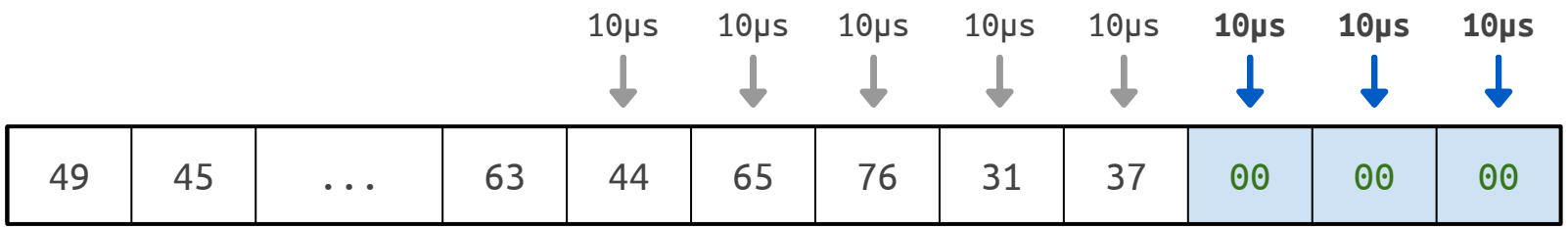
```
int32_t remove_padding3(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    if (i >= buflen - padlen) {
      if (b != padlen)
        ok = 0;
      b = 0;
    }
    buf[i] = b;
  }
  return ok ? padlen : -1;
}
```



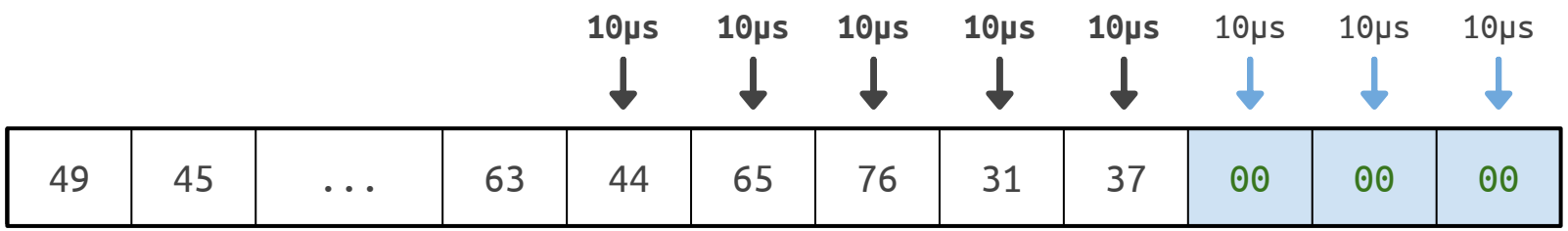
```c
int32_t remove_padding3(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    if (i >= buflen - padlen) {
      if (b != padlen)
        ok = 0;
      b = 0;
    }
    buf[i] = b;
  }
  return ok ? padlen : -1;
}
```



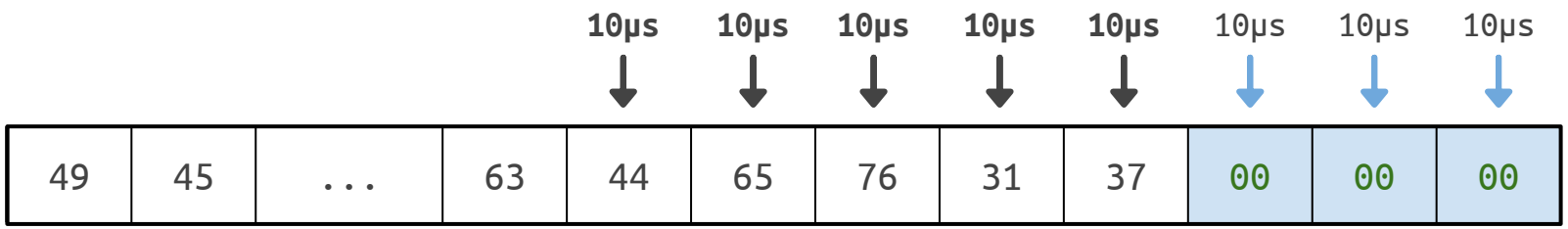
```
int32_t remove_padding3(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    if (i >= buflen - padlen) {
      if (b != padlen)
        ok = 0;
      b = 0;
    }
    buf[i] = b;
  }
  return ok ? padlen : -1;
}
```



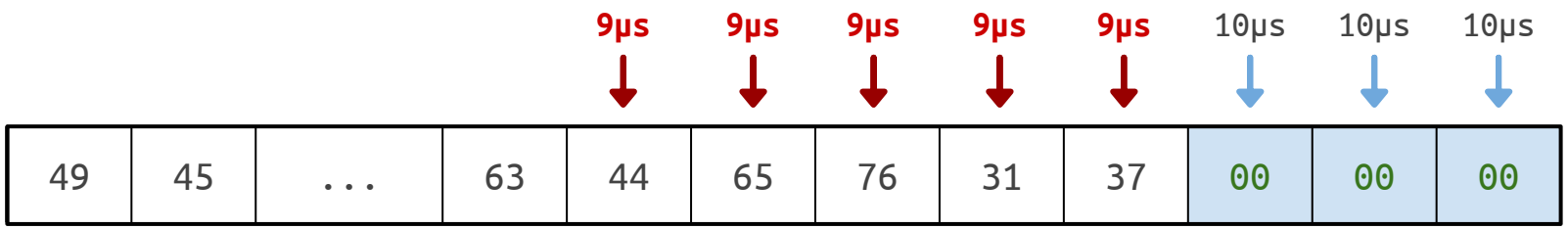
```c
int32_t remove_padding3(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    if (i >= buflen - padlen) {
      if (b != padlen)
        ok = 0;
      b = 0;
    }
    buf[i] = b;
  }
  return ok ? padlen : -1;
}
```



```
int32_t remove_padding3(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    if (i >= buflen - padlen) {
      if (b != padlen)
        ok = 0;
      b = 0;
    }
    buf[i] = b;
  }
  return ok ? padlen : -1;
}
```

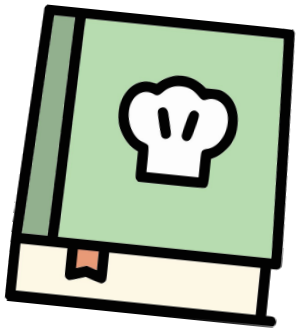


```
int32_t remove_padding3(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    if (i >= buflen - padlen) {
      if (b != padlen)
        ok = 0;
      b = 0;
    }
    buf[i] = b;
  }
  return ok ? padlen : -1;
}
```

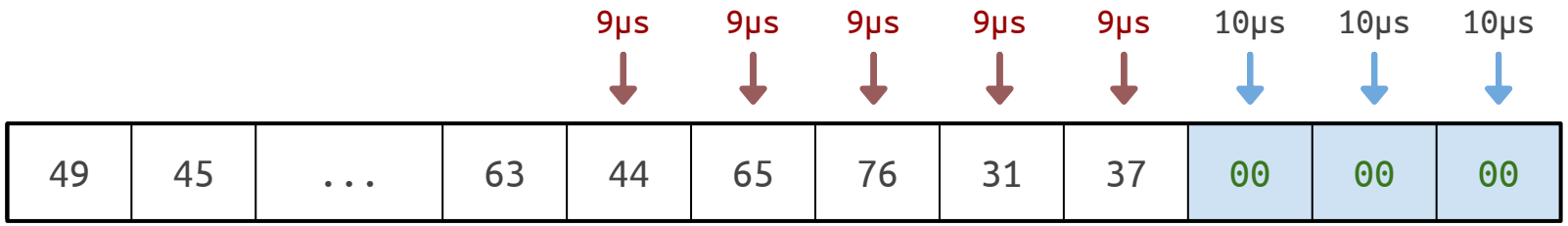


```c
int32_t remove_padding3(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    if (i >= buflen - padlen) {
      if (b != padlen)
        ok = 0;
      b = 0;
    }
    buf[i] = b;
  }
  return ok ? padlen : -1;
}
```

It's dangerous to have branching code!



Use this instead.



```c
int32_t remove_padding3(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    if (i >= buflen - padlen) {
      if (b != padlen)
        ok = 0;
      b = 0;
    }
    buf[i] = b;
  }
  return ok ? padlen : -1;
}
```

Ugly! Do not read!

```c
int32_t remove_padding4(
    uint8_t* buf,
    uint32_t buflen) {
  uint32_t ok = -1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    uint32_t proper_index =
      ct_ge_u32(i, buflen - padlen);
    uint32_t matches_pad =
      ct_eq_u8(b, padlen);
    ok &= matches_pad & proper_index;
    b = ~proper_index & b;
    buf[i] = b;
  }
  return (ok & padlen) | ~ok;
}
```

9µs	9µs	9µs	9µs	9µs	10µs	10µs	10µs
44	65	76	31	37	00	00	00

49	45	...	63

```c
int32_t remove_padding3(
    uint8_t* buf,
    uint32_t buflen) {
  uint8_t ok = 1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    if (i >= buflen - padlen) {
      if (b != padlen)
        ok = 0;
      b = 0;
    }
    buf[i] = b;
  }
  return ok ? padlen : -1;
}
```

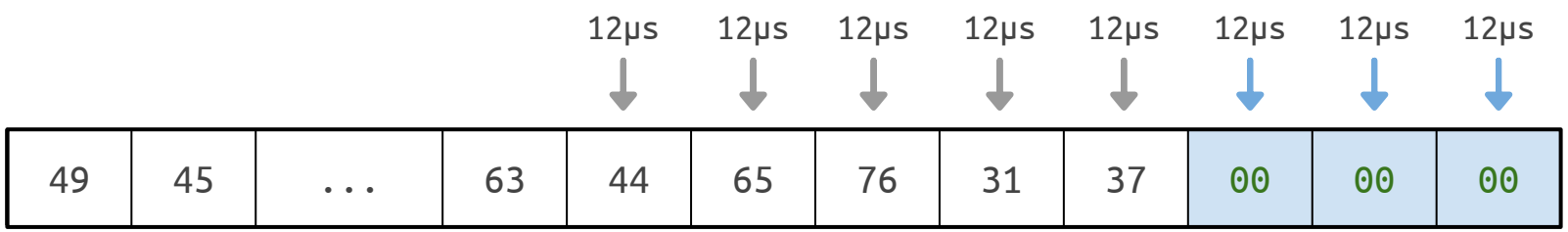
Ugly! Do not read!

```c
int32_t remove_padding4(
    uint8_t* buf,
    uint32_t buflen) {
  uint32_t ok = -1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    uint32_t proper_index =
      ct_ge_u32(i, buflen - padlen);
    uint32_t matches_pad =
      ct_eq_u8(b, padlen);
    ok &= matches_pad & proper_index;
    b = ~proper_index & b;
    buf[i] = b;
  }
  return (ok & padlen) | ~ok;
}
```

12µs	12µs	12µs	12µs	12µs	12µs	12µs	12µs
49	45	...	63	44	65	76	31	37	00	00	00

```
int32_t remove_padding4(
    uint8_t* buf,
    uint32_t buflen) {
  uint32_t ok = -1;
  uint8_t padlen = buf[buflen-1];
  uint32_t i;
  for (i = buflen-255; i < buflen; i++) {
    uint8_t b = buf[i];
    uint32_t proper_index = ct_ge_u32(i, buflen - padlen);
    uint32_t matches_pad = ct_eq_u8(b, padlen);
    ok &= matches_pad & proper_index;
    b = ~proper_index & b;
    buf[i] = b;
  }
  return (ok & padlen) | ~ok;
}
```

Ugly! Do not read!



Error-prone in practice

```diff
@@ -383,13 +383,10 @@ dtls1_process_record(SSL *s)
 383  383        SSL3_RECORD *rr;
 384  384        unsigned int mac_size;
 385  385        unsigned char md[EVP_MAX_MD_SIZE];
      386 +      int decryption_failed_or_bad_record_mac = 0;
 386  387
 387  388
 388  389        rr= &(s->s3->rrec);
@@ -417,13 +418,10 @@ dtls1_process_record(SSL *s)
 417  418        enc_err = s->method->ssl3_enc->enc(s,0);
 418  419        if (enc_err <= 0)
 419  420                {
 420     -              /* decryption failed, silently discard message */
 421     -              if (enc_err < 0)
 422     -                      {
 423     -                      rr->length = 0;
 424     -                      s->packet_length = 0;
 425     -                      }
 426     -              goto err;
      421 +              /* To minimize information leaked via timing, we will always
      422 +               * perform all computations before discarding the message.
      423 +               */
      424 +              decryption_failed_or_bad_record_mac = 1;
 427  425                }
 428  426
 429  427 #ifdef TLS_DEBUG
@@ -453,7 +451,7 @@ printf("\n");
 453  451                        SSLerr(SSL_F_DTLS1_PROCESS_RECORD,SSL_R_PRE_MAC_LENGTH_TOO_LONG);
 454  452                        goto f_err;
 455  453 #else
 456     -                      goto err;
      454 +                      decryption_failed_or_bad_record_mac = 1;
 457  455 #endif
 458  456                        }
 459  457                /* check the MAC for rr->input (it's in mac_size bytes at the tail) */
@@ -464,17 +462,25 @@ printf("\n");
 464  462                        SSLerr(SSL_F_DTLS1_PROCESS_RECORD,SSL_R_LENGTH_TOO_SHORT);
 465  463                        goto f_err;
 466  464 #else
 467     -                      goto err;
      465 +                      decryption_failed_or_bad_record_mac = 1;
 468  466 #endif
 469  467                        }
 470  468                rr->length-=mac_size;
 471  469                i=s->method->ssl3_enc->mac(s,md,0);
 472  470                if (i < 0 || memcmp(md,&(rr->data[rr->length]),mac_size) != 0)
 473  471                        {
 474     -                      goto err;
      472 +                      decryption_failed_or_bad_record_mac = 1;
 475  473                        }
 476  474                }
 477  475
      476 +      if (decryption_failed_or_bad_record_mac)
      477 +              {
      478 +              /* decryption failed, silently discard message */
      479 +              rr->length = 0;
      480 +              s->packet_length = 0;
      481 +              goto err;
      482 +              }
      483 +
 478  484        /* r->length is now just compressed */
 479  485        if (s->expand != NULL)
 480  486                {
```

OpenSSL padding oracle attack

Canvel, et al. “Password Interception in a SSL/TLS Channel.” *Crypto,* Vol. 2729. 2003.

Error-prone in practice

```
-       EVP_DigestUpdate(&md_ctx,md,2);
-       EVP_DigestUpdate(&md_ctx,rec->input,rec->length);
-       EVP_DigestFinal_ex( &md_ctx,md,NULL);
-
-       EVP_MD_CTX_copy_ex( &md_ctx,hash);
-       EVP_DigestUpdate(&md_ctx,mac_sec,md_size);
-       EVP_DigestUpdate(&md_ctx,ssl3_pad_2,npad);
-       EVP_DigestUpdate(&md_ctx,md,md_size);
-       EVP_DigestFinal_ex( &md_ctx,md,&md_size);
-
-       EVP_MD_CTX_cleanup(&md_ctx);
+       if (!send &&
+           EVP_CIPHER_CTX_mode(ssl->enc_read_ctx) == EVP_CIPH_CBC_MODE &&
+           ssl3_cbc_record_digest_supported(hash))
+               {
+               /* This is a CBC-encrypted record. We must avoid leaking any
+                * timing-side channel information about how many blocks of
+                * data we are hashing because that gives an attacker a
+                * timing-oracle. */
+
+               /* npad is, at most, 48 bytes and that's with MD5:
+                *   16 + 48 + 8 (sequence bytes) + 1 + 2 = 75.
+                *
+                * With SHA-1 (the largest hash speced for SSLv3) the hash size
+                * goes up 4, but npad goes down by 8, resulting in a smaller
+                * total size. */
+               unsigned char header[75];
+               unsigned j = 0;
+               memcpy(header+j, mac_sec, md_size);
+               j += md_size;
+               memcpy(header+j, ssl3_pad_1, npad);
+               j += npad;
+               memcpy(header+j, seq, 8);
+               j += 8;
+               header[j++] = rec->type;
+               header[j++] = rec->length >> 8;
+               header[j++] = rec->length & 0xff;
+
+               ssl3_cbc_digest_record(
+                       hash,
+                       md, &md_size,
+                       header, rec->input,
+                       rec->length + md_size, rec->orig_len,
+                       mac_sec, md_size,
+                       1 /* is SSLv3 */);
+               }
+       else
+               {
+               unsigned int md_size_u;
+               /* Chop the digest off the end :-) */
+               EVP_MD_CTX_init(&md_ctx);
+
+               EVP_MD_CTX_copy_ex( &md_ctx,hash);
+               EVP_DigestUpdate(&md_ctx,mac_sec,md_size);
+               EVP_DigestUpdate(&md_ctx,ssl3_pad_1,npad);
+               EVP_DigestUpdate(&md_ctx,seq,8);
+               rec_char=rec->type;
+               EVP_DigestUpdate(&md_ctx,&rec_char,1);
+               p=md;
+               s2n(rec->length,p);
+               EVP_DigestUpdate(&md_ctx,md,2);
+               EVP_DigestUpdate(&md_ctx,rec->input,rec->length);
```

Lucky 13 timing attack

Al Fardan and Paterson. “Lucky thirteen: Breaking the TLS and DTLS record protocols.” Oakland 2013.

Error-prone in practice

```
-       EVP_DigestUpdate(&md_ctx,md,2);
-       EVP_DigestUpdate(&md_ctx,rec->input,rec->length);
-       EVP_DigestFin
-
-       EVP_MD_CTX_co
-       EVP_DigestUpd
-       EVP_DigestUpd
-       EVP_DigestUpd
-       EVP_DigestFin
-
-       EVP_MD_CTX_cl
+       if (!send &&
+           EVP_CIPHE
+           ssl3_cbc_
+               {
+               /* Th
+                * ts
+                * da
+                * ti
+
+               /* np
+                *
+                *
+                * Wh
+                * go
+                * to
+               unsig
+               unsig
+               memcp
+               j +=
+               memcp
+               j +=
+               memcp
+               j +=
+               heade
+               heade
+               heade
+
+               ssl3_
+
+
+
+
+
+
+               }
+       else
+               {
+               unsig
+               /* Ch
+               EVP_M
+
+               EVP_M
+               EVP_D
+               EVP_D
+               EVP_D
+               rec_c
+               EVP_D
+               p=md;
+               s2n(r
+               EVP_D
+               EVP_D
```

```
-               unsigned char mac[SHA_DIGEST_LENGTH];
+               union { unsigned int  u[SHA_DIGEST_LENGTH/sizeof(unsigned int)];
+                       unsigned char c[SHA_DIGEST_LENGTH]; } mac;

                /* decrypt HMAC|padding at once */
                aesni_cbc_encrypt(in,out,len,
                                &key->ks,ctx->iv,0);

                if (plen) {     /* "TLS" mode of operation */
-                       /* figure out payload length */
-                       if (len<(size_t)(out[len-1]+1+SHA_DIGEST_LENGTH))
-                               return 0;
-
-                       len -= (out[len-1]+1+SHA_DIGEST_LENGTH);
+                       size_t inp_len, mask, j, i;
+                       unsigned int res, maxpad, pad, bitlen;
+                       int ret = 1;
+                       union { unsigned int  u[SHA_LBLOCK];
+                               unsigned char c[SHA_CBLOCK]; }
+                               *data = (void *)key->md.data;

                        if ((key->aux.tls_aad[plen-4]<<8|key->aux.tls_aad[plen-3])
-                           >= TLS1_1_VERSION) {
-                               len -= AES_BLOCK_SIZE;
+                           >= TLS1_1_VERSION)
                                iv = AES_BLOCK_SIZE;
-                       }

-                       key->aux.tls_aad[plen-2] = len>>8;
-                       key->aux.tls_aad[plen-1] = len;
+                       if (len<(iv+SHA_DIGEST_LENGTH+1))
+                               return 0;
+
+                       /* omit explicit iv */
+                       out += iv;
+                       len -= iv;
+
+                       /* figure out payload length */
+                       pad = out[len-1];
+                       maxpad = len-(SHA_DIGEST_LENGTH+1);
+                       maxpad |= (255-maxpad)>>(sizeof(maxpad)*8-8);
+                       maxpad &= 255;
+
+                       inp_len = len - (SHA_DIGEST_LENGTH+pad+1);
+                       mask = (0-((inp_len-len)>>(sizeof(inp_len)*8-1)));
+                       inp_len &= mask;
+                       ret &= (int)mask;

-                       /* calculate HMAC and verify it */
+                       key->aux.tls_aad[plen-2] = inp_len>>8;
+                       key->aux.tls_aad[plen-1] = inp_len;
+
+                       /* calculate HMAC */
                        key->md = key->head;
                        SHA1_Update(&key->md,key->aux.tls_aad,plen);
-                       SHA1_Update(&key->md,out+iv,len);
-                       SHA1_Final(mac,&key->md);

+#if 1
+                       len -= SHA_DIGEST_LENGTH;               /* amend mac */
+                       if (len>=(256+SHA_CBLOCK)) {
+                               j = (len-(256+SHA_CBLOCK))&(0-SHA_CBLOCK);
+                               j += SHA_CBLOCK-key->md.num;
```

Further refinements

Decryption path has no more measurable timing differences

Error-prone in practice

```
-       EVP_DigestUpdate(&md_ctx,md,2);
-       EVP_DigestUpdate(&md_ctx,rec->input,rec->length);
-       EVP_DigestFin
-
-       EVP_MD_CTX_co
-       EVP_DigestUpd
-       EVP_DigestUpd
-       EVP_DigestUpd
-       EVP_DigestFin
-
-       EVP_MD_CTX_cl
+       if (!send &&
+           EVP_CIPHE
+           ssl3_cbc_
+               {
+               /* Th
+                * ts
+                * da
+                * ti
+
+               /* np
+                *
+                *
+                * Wh
+                * go
+                * to
+               unsig
+               unsig
+               memcp
+               j +=
+               memcp
+               j +=
+               memcp
+               j +=
+               heade
+               heade
+               heade
+
+               ssl3_
+
+
+
+
+
+
+               }
+       else
+               {
+               unsig
+               /* Ch
+               EVP_M
+
+               EVP_M
+               EVP_D
+               EVP_D
+               EVP_D
+               rec_c
+               EVP_D
+               p=md;
+               s2n(r
+               EVP_D
+               EVP_D
```

```
-               unsigned char mac[SHA_DIGEST_LENGTH];
+               union { unsigned int  u[SHA_DIGEST_LENGTH/sizeof(unsigned int)];
+                       unsigned char c[SHA_DIGEST_LENGTH]; } mac;

                /* decrypt HMAC|padding at once */
                aesni_cbc_encrypt(in,out,len,
                                &key->ks,ctx->iv,0);

                if (plen) {     /* "TLS" mode of operation */
-                       /* figure out payload length */
-                       if (len<(size_t)(out[len-1]+1+SHA_DIGEST_LENGTH))
-                               return 0;
-
-                       len -= (out[len-1]+1+SHA_DIGEST_LENGTH);
+                       size_t inp_len, mask, j, i;
+                       unsigned int res, maxpad, pad, bitlen;
+                       int ret = 1;
+                       union { unsigned int  u[SHA_LBLOCK];
+                               unsigned char c[SHA_CBLOCK]; }
+                               *data = (void *)key->md.data;

                        if ((key->aux.tls_aad[plen-4]<<8|key->aux.tls_aad[plen-3])
-                           >= TLS1_1_VERSION) {
-                               len -= AES_BLOCK_SIZE;
+                           >= TLS1_1_VERSION)
                                iv = AES_BLOCK_SIZE;
-                       }

-                       key->aux.tls_aad[plen-2] = len>>8;
-                       key->aux.tls_aad[plen-1] = len;
+                       if (len<(iv+SHA_DIGEST_LENGTH+1))
+                               return 0;
+
+                       /* omit explicit iv */
+                       out += iv;
+                       len -= iv;
+
+                       /* figure out payload length */
+                       pad = out[len-1];
+                       maxpad = len-(SHA_DIGEST_LENGTH+1);
+                       maxpad |= (255-maxpad)>>(sizeof(maxpad)*8-8);
+                       maxpad &= 255;
+
+                       inp_len = len - (SHA_DIGEST_LENGTH+pad+1);
+                       mask = (0-((inp_len-len)>>(sizeof(inp_len)*8-1)));
+                       inp_len &= mask;
+                       ret &= (int)mask;

-                       /* calculate HMAC and verify it */
+                       key->aux.tls_aad[plen-2] = inp_len>>8;
+                       key->aux.tls_aad[plen-1] = inp_len;
+
+                       /* calculate HMAC */
                        key->md = key->head;
                        SHA1_Update(&key->md,key->aux.tls_aad,plen);
-                       SHA1_Update(&key->md,out+iv,len);
-                       SHA1_Final(mac,&key->md);

+#if 1
+                       len -= SHA_DIGEST_LENGTH;               /* amend mac */
+                       if (len>=(256+SHA_CBLOCK)) {
+                               j = (len-(256+SHA_CBLOCK))&(0-SHA_CBLOCK);
+                               j += SHA_CBLOCK-key->md.num;
```

Further refinements

Decryption path has no more measurable timing differences

Error-prone in practice

```
583   584          maxpad |= (255 - maxpad) >> (sizeof(maxpad) * 8 - 8);
584   585          maxpad &= 255;
585   586
      587    +     ret &= constant_time_ge(maxpad, pad);
      588    +
586   589          inp_len = len - (SHA_DIGEST_LENGTH + pad + 1);
587   590          mask = (0 - ((inp_len - len) >> (sizeof(inp_len) * 8 - 1)));
588   591          inp_len &= mask;
```

CVE-2016-2107

Somorovsky. “Curious padding oracle in OpenSSL.”

Writing constant-time code is hard

- **Challenge**: manually ensuring code is CT
  - Standard programming constructs introduce timing leaks
  - Manually keep track of secret vs. public

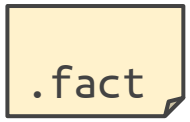
- **Consequence**: vulnerabilities!
  - Difficult to write correct code
  - Hard to understand what CT code is doing
  - Hard to maintain CT code

This is what DSLs are for!

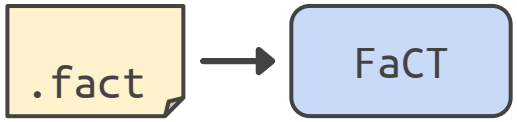
- Explicitly distinguish secret vs. public values
- Type system to prevent writing leaky code
- Compiler to transform high-level constructs to CT

FaCT

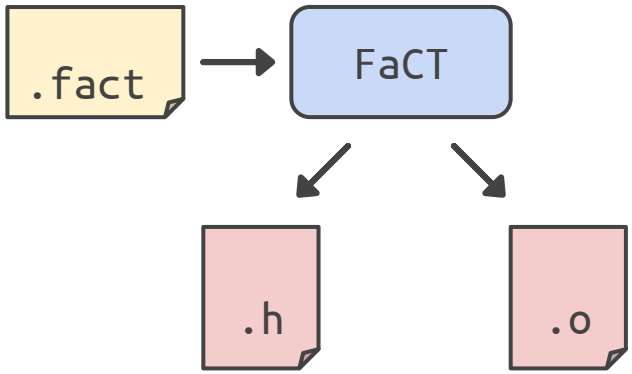
FaCT



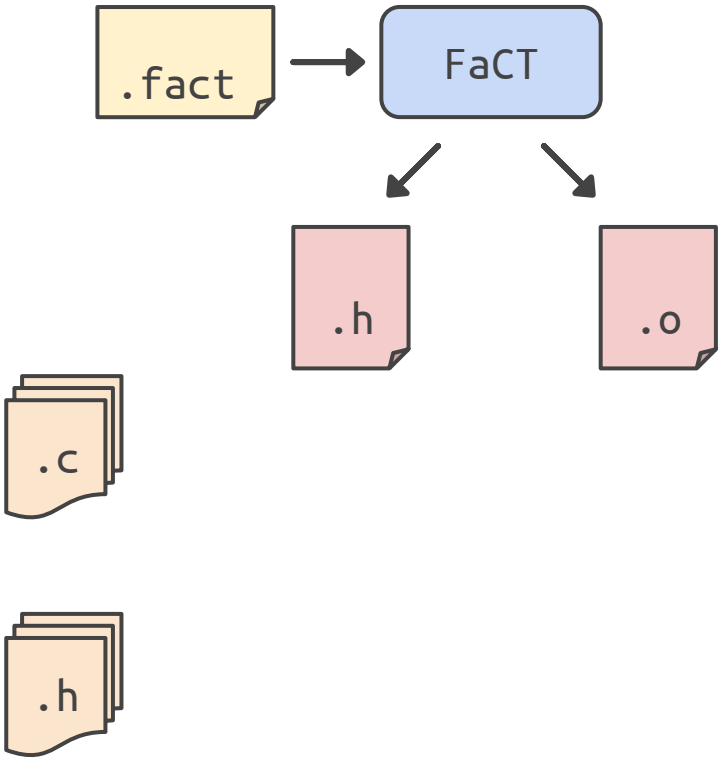
FaCT



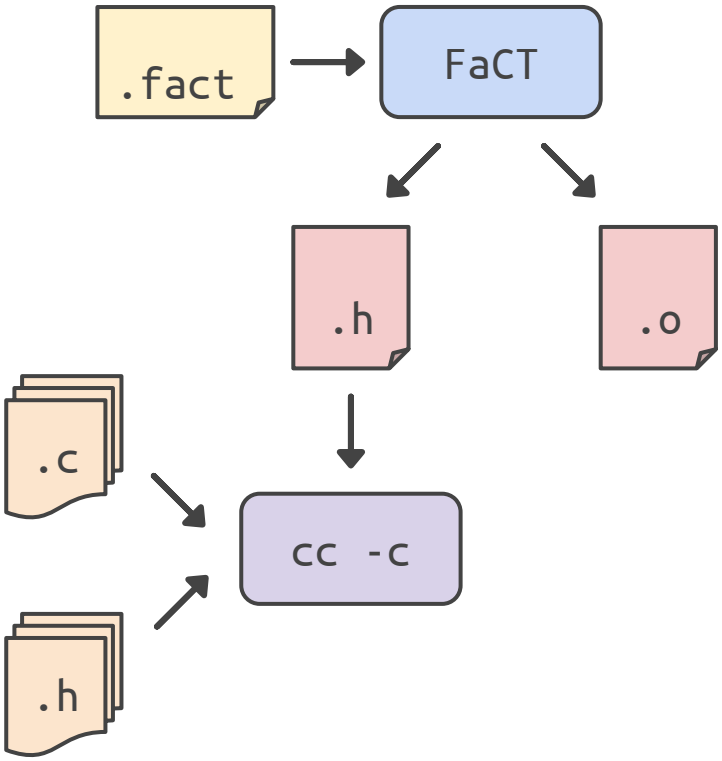
FaCT



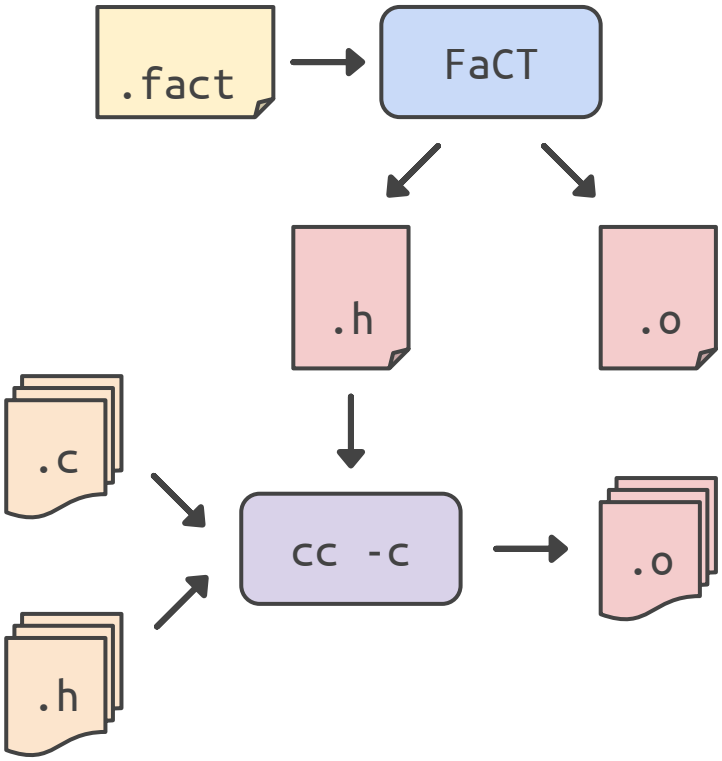
FaCT



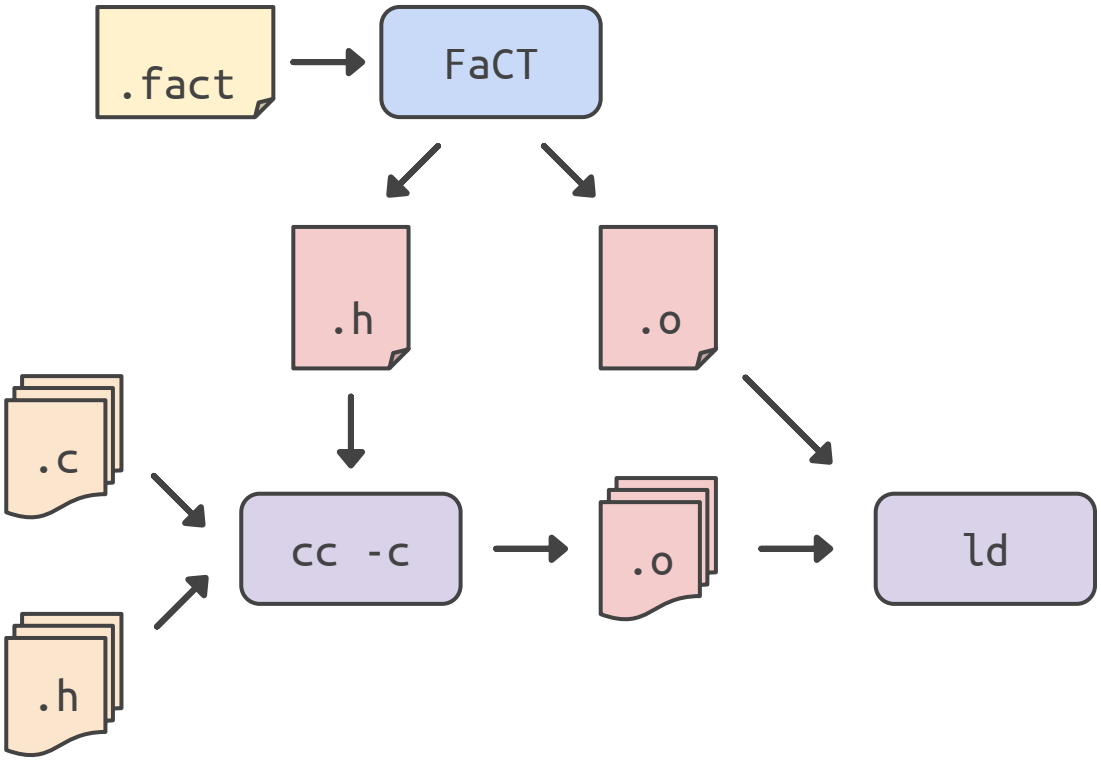
FaCT



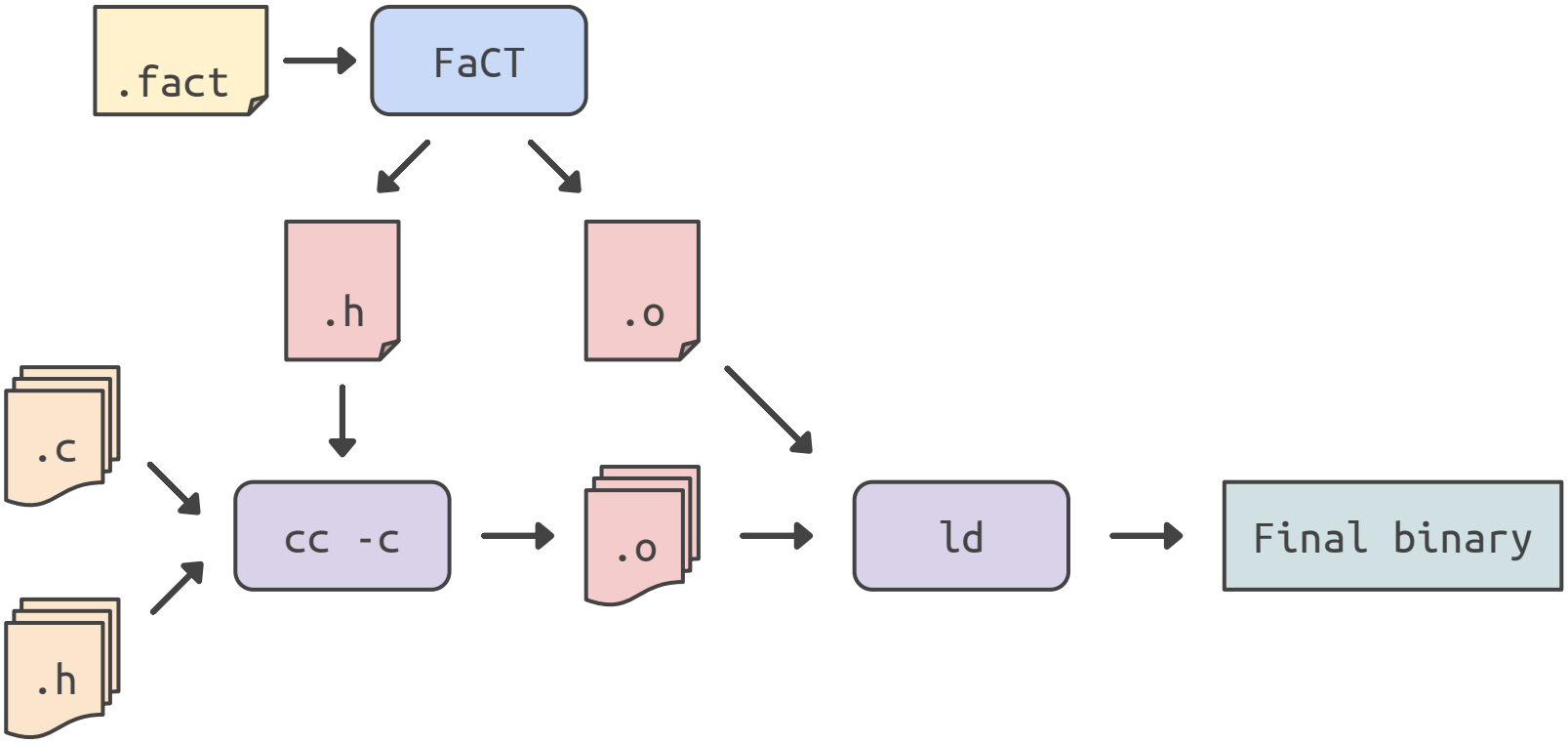
FaCT



FaCT



FaCT



What does FaCT look like?

```
secret int32 remove_padding(secret mut uint8[] buf) {
  assume(len buf >= 255);
  secret uint8 padlen = buf[len buf - 1];
  for (uint64 i from len buf - 255 to len buf) {
    if (i >= len buf - padlen) {
      if (buf[i] != padlen) {
        return -1;
      }
      buf[i] = 0;
    }
  }
  return int32(padlen);
}
```

What does FaCT look like?

```
secret int32 remove_padding(secret mut uint8[] buf) {
  assume(len buf >= 255);
  secret uint8 padlen = buf[len buf - 1];
  for (uint64 i from len buf - 255 to len buf) {
    if (i >= len buf - padlen) {
      if (buf[i] != padlen) {
        return -1;
      }
      buf[i] = 0;
    }
  }
  return int32(padlen);
}
```

What does FaCT look like?

```
secret int32 remove_padding(secret mut uint8[] buf) {
  assume(len buf >= 255);
  secret uint8 padlen = buf[len buf - 1];
  for (uint64 i from len buf - 255 to len buf) {
    if (i >= len buf - padlen) {
      if (buf[i] != padlen) {
        return -1;
      }
      buf[i] = 0;
    }
  }
  return int32(padlen);
}
```

What does FaCT look like?

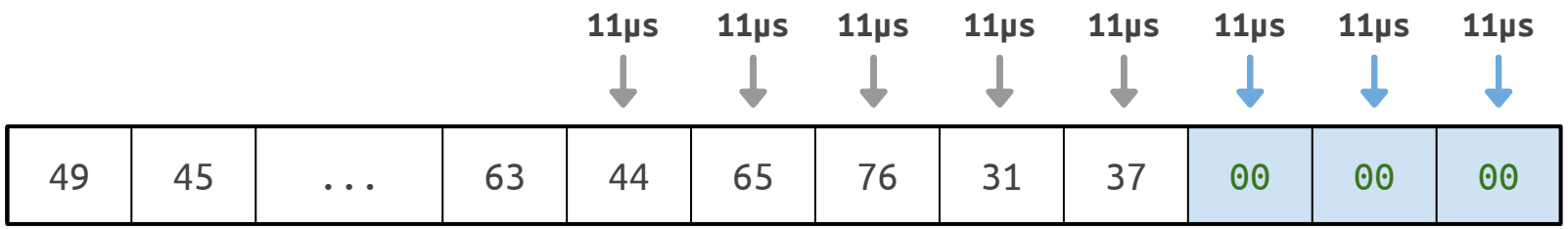
```
secret int32 remove_padding(secret mut uint8[] buf) {
  assume(len buf >= 255);
  secret uint8 padlen = buf[len buf - 1];
  for (uint64 i from len buf - 255 to len buf) {
    if (i >= len buf - padlen) {
      if (buf[i] != padlen) {
        return -1;
      }
      buf[i] = 0;
    }
  }
  return int32(padlen);
}
```

What does FaCT look like?

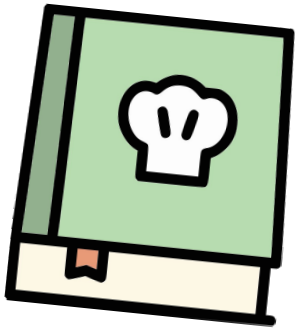
```
secret int32 remove_padding(secret mut uint8[] buf) {
  assume(len buf >= 255);
  secret uint8 padlen = buf[len buf - 1];
  for (uint64 i from len buf - 255 to len buf) {
    if (i >= len buf - padlen) {
      if (buf[i] != padlen) {
        return -1;
      }
      buf[i] = 0;
    }
  }
  return int32(padlen);
}
```

What does FaCT look like?

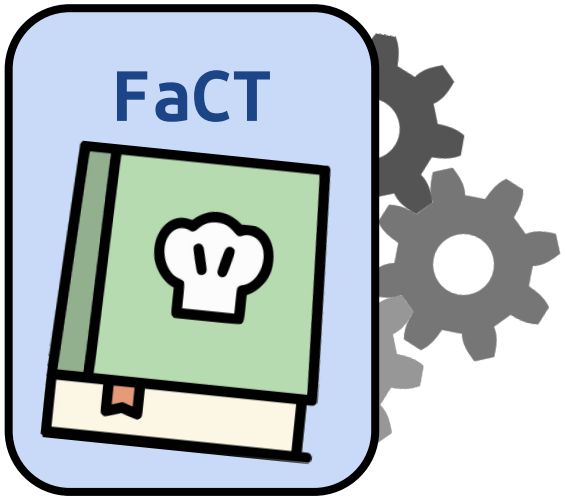
```
secret int32 remove_padding(secret mut uint8[] buf) {
  assume(len buf >= 255);
  secret uint8 padlen = buf[len buf - 1];
  for (uint64 i from len buf - 255 to len buf) {
    if (i >= len buf - padlen) {
      if (buf[i] != padlen) {
        return -1;
      }
      buf[i] = 0;
    }
  }
  return int32(padlen);
}
```



Automatically transform code



Automatically transform code



Automatically transform code

- Transform secret branches into straight-line code
- Keep track of static control flow

```
if (s) {
    if (s2) {
        x = 42;
    } else {
        x = 17;
    }
    y = x + 2;
}
```

Automatically transform code

- Transform secret branches into straight-line code
- Keep track of static control flow

```
if (s) {
    if (s2) {
        x = 42;
    } else {
        x = 17;
    }
    y = x + 2;
}
```

```
x = ct_select(s & s2, 42, x);
x = ct_select(s & ~s2, 17, x);
y = ct_select(s, x + 2, y);
```

Automatically transform code

- Transform away early returns
- Keep track of current return state

```
if (s) {
    return 42;
}
return 17;
```

Automatically transform code

- Transform away early returns
- Keep track of current return state

```
if (s) {
    return 42;
}
return 17;
```

```
rval = ct_select(s & ~returned, 42, rval);
returned |= s;

rval = ct_select(~returned, 17, rval);
returned |= true;

      ⋮

return rval;
```

Automatically transform code

- Transform function side effects
  - Depends on control flow state of caller
- Pass the current control flow as an extra parameter

```
void fn(mut x) {
    x = 42;
}


if (s) {
    fn(ref x);
}
```

Automatically transform code

- Transform function side effects
  - Depends on control flow state of caller
- Pass the current control flow as an extra parameter

```
void fn(mut x) {
    x = 42;
}
```

```
void fn(mut x, bool state) {
    x = ct_select(state, 42, x);
}
```

```
if (s) {
    fn(ref x);
}
```

```
fn(ref x, s);
```

Labels ensure no leakage

- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions we can’t transform:

Labels ensure no leakage

- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions we can't transform:
  - Loop bounds

```
for (uint32 i from 0 to secret_value) {
    do_operation();
}
```

Labels ensure no leakage

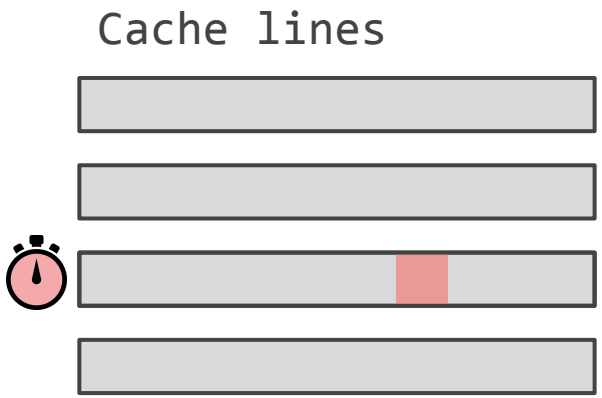
- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions we can't transform:
  - Loop bounds

```
for (uint32 i from 0 to public_value) {
    if (i < secret_value) {
        do_operation();
    }
}
```

Labels ensure no leakage

- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions we can't transform:
  - Loop bounds
  - Array indices

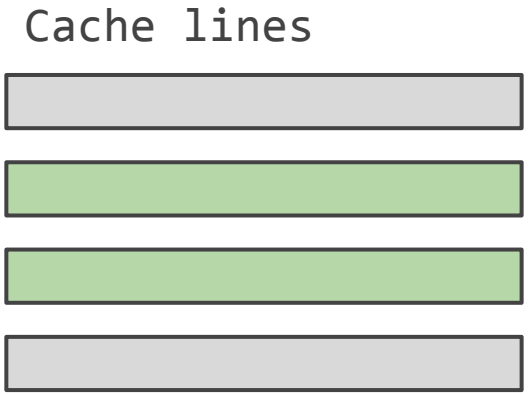
```
x = sensitive_buffer[secret_value];
```



Labels ensure no leakage

- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions we can't transform:
  - Loop bounds
  - Array indices

```
for (uint32 i from public_lo to public_hi) {
    if (i == secret_value) {
        x = sensitive_buffer[i];
    }
}
```



Labels ensure no leakage

- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions we can't transform:
  - Loop bounds
  - Array indices
  - Variable-time instructions

```
x = public_value / secret_value2;
```

Labels ensure no leakage

- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions we can't transform:
  - Loop bounds
  - Array indices
  - Variable-time instructions

```
x = public_value / public_value2;
```

OR

```
x = ct_div(public_value, secret_value2);
```

Labels ensure no leakage

- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions we can't transform:
  - Loop bounds
  - Array indices
  - Variable-time instructions
  - Recursive calls

```
secret uint32 fact(secret uint32 n) {
  if (n > 1) {
    return n * fact(n - 1);
  }
  return 1;
}
```

Labels ensure no leakage

- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions we can't transform:
  - Loop bounds
  - Array indices
  - Variable-time instructions
  - Recursive calls

```
secret uint32 fact(secret uint32 n) {
  if (n > 1) {
    return n * fact(n - 1);
  }
  return 1;
}
```

Transformations are tricky

```
if (secret_modeval == HAS_32_BYTES) {
  buf[31] = 0;
}
```

Transformations are tricky

```
if (secret_modeval == HAS_32_BYTES) {
  buf[31] = 0;
}
```

```
m = (secret_modeval == HAS_32_BYTES);
buf[31] = ct_select(m, 0, buf[31]);
```

Transformations are tricky

```
if (secret_modeval == HAS_32_BYTES) {
  buf[31] = 0;
}
```

```
m = (secret_modeval == HAS_32_BYTES);
buf[31] = ct_select(m, 0, buf[31]);
```

- **Problem:** secret if-statements always perform branches
  - Does not guard execution
  - Similar problem for secret early return

Transformations are tricky

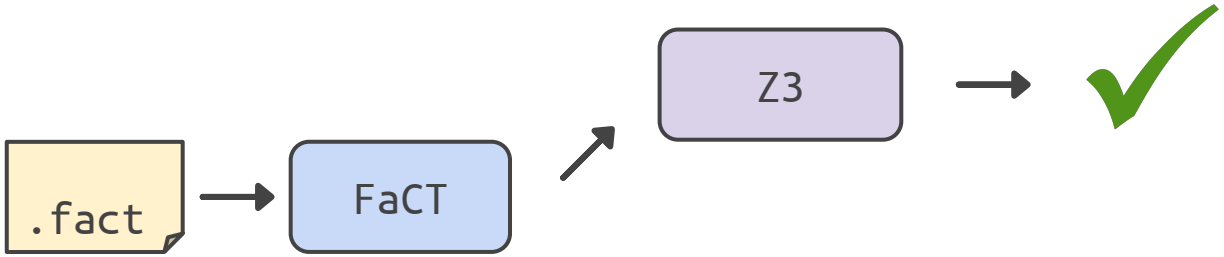
```
if (secret_modeval == HAS_32_BYTES) {
  buf[31] = 0;
}
```

```
m = (secret_modeval == HAS_32_BYTES);
buf[31] = ct_select(m, 0, buf[31]);
```

- **Problem:** secret if-statements always perform branches
  - Does not guard execution
  - Similar problem for secret early return
- **Solution:** disallow these programs!

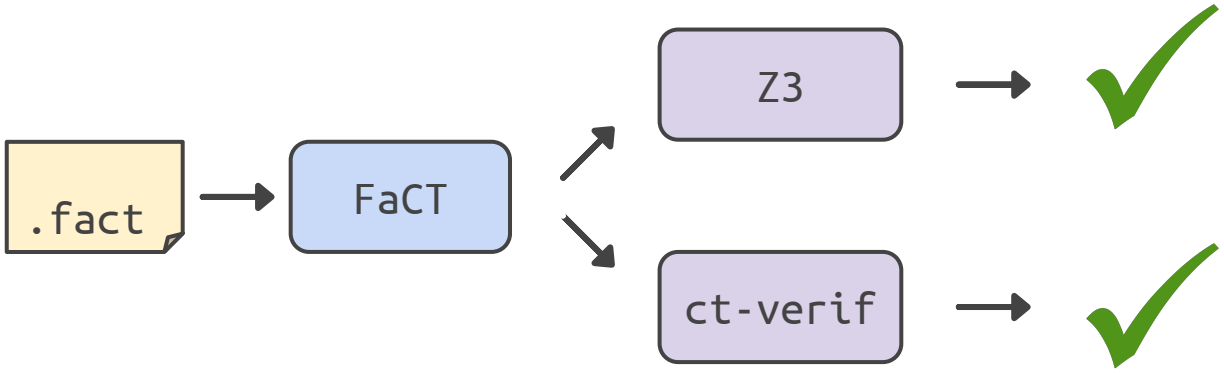
Verifying constant-time code

- FaCT is memory safe and has no undefined behavior
  - Generate constraints while type checking
  - Aware of secret-if semantics



Verifying constant-time code

- FaCT is memory safe and has no undefined behavior
  - Generate constraints while type checking
  - Aware of secret-if semantics
- FaCT generates constant-time code
  - Verified with external tool



Ok, but how good is it really?

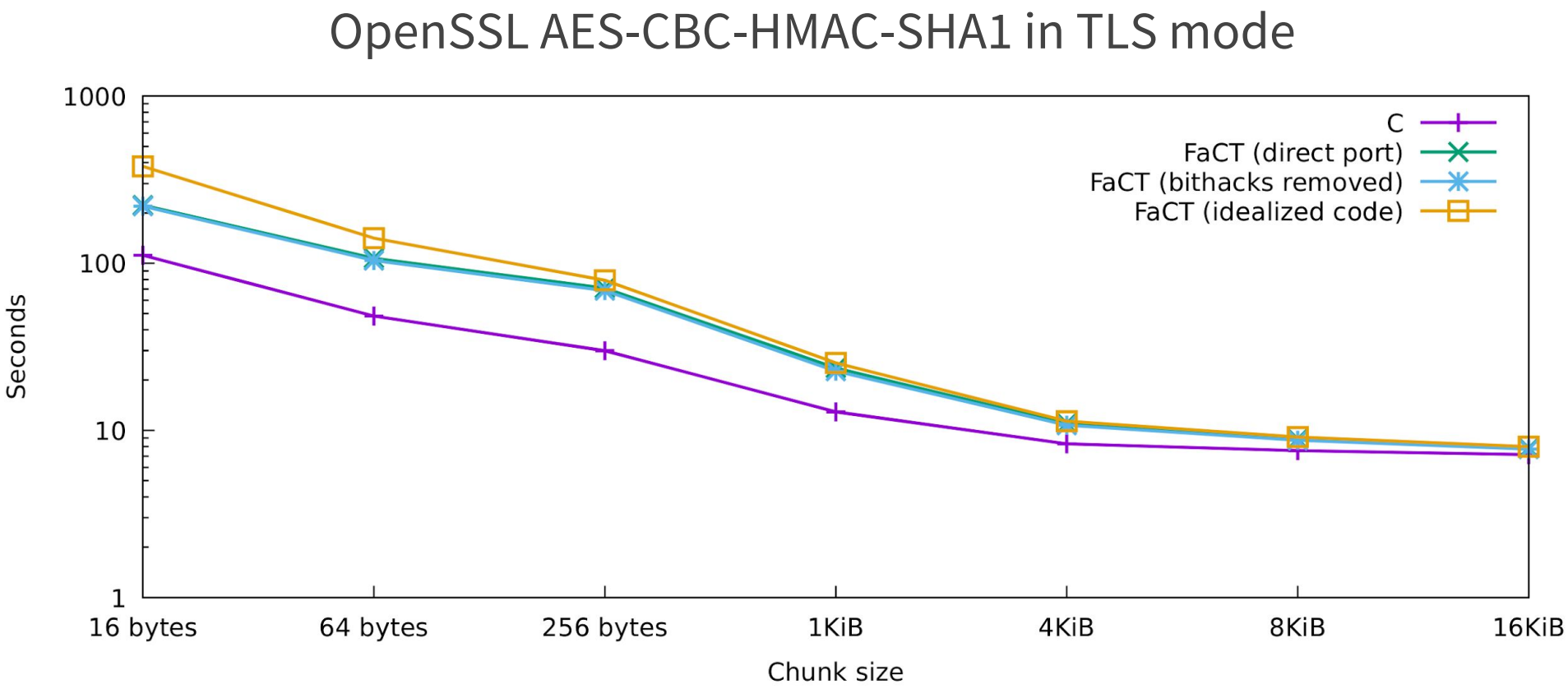
- How fast is it?
- How usable is it?

Ok, but how good is it really?

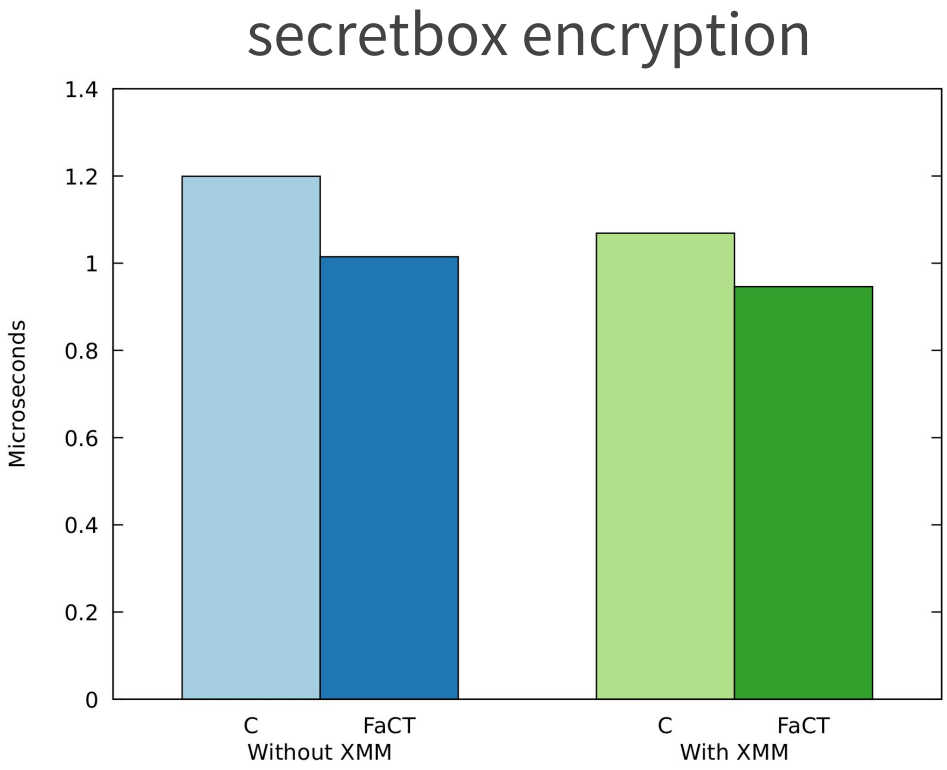
- How fast is it?
- How usable is it?

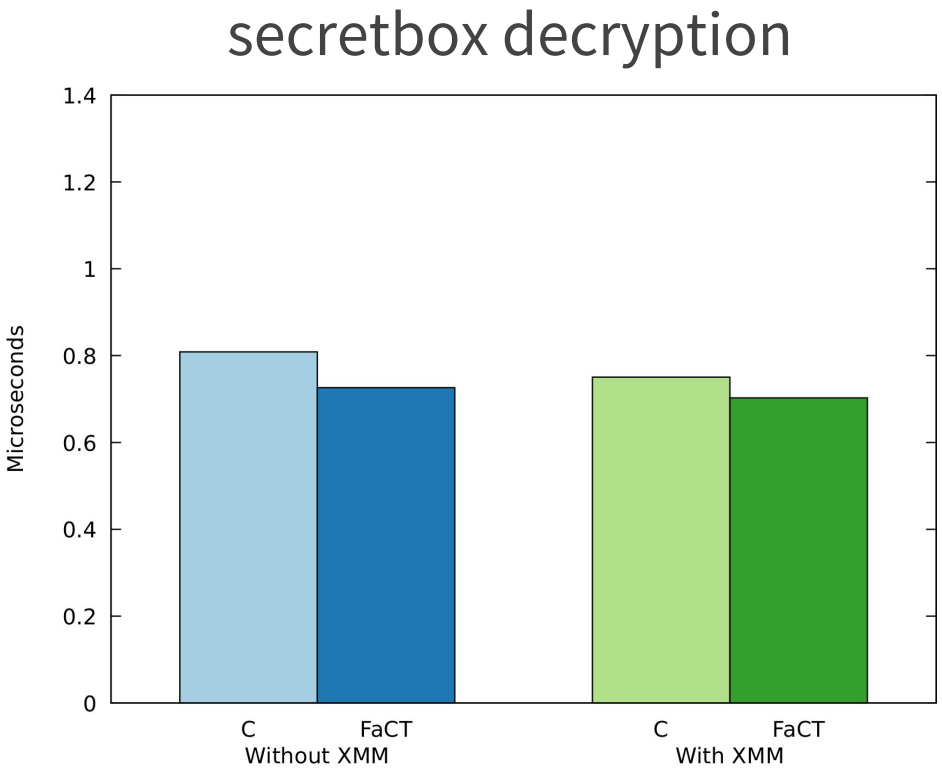
- Case studies:
  - libsodium's crypto_secretbox
  - OpenSSL's AES-CBC-HMAC-SHA1
  - mbedTLS's Montgomery multiplication & sliding window exponentiation
  - DJB/Langley's curve-25519

Performance numbers



Performance numbers





Future directions

- Optimize transformations
- Add other backends (ARM, CT-WASM, ...)
- Verify the FaCT compiler

FaCT https://github.com/PLSysSec/FaCT

- DSL for cryptographic code
- Automatic transformation to constant-time
- Easily fits into your existing toolchain
- Usable and fast