

# Constant-time programming in FaCT

**Sunjay Cauligi**, UC San Diego

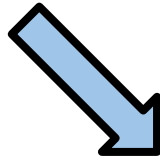
Fraser Brown, Ranjit Jhala, Brian Johannesmeyer,  
John Renner, Gary Soeller, Deian Stefan, Riad Wahby

# Timing side channels

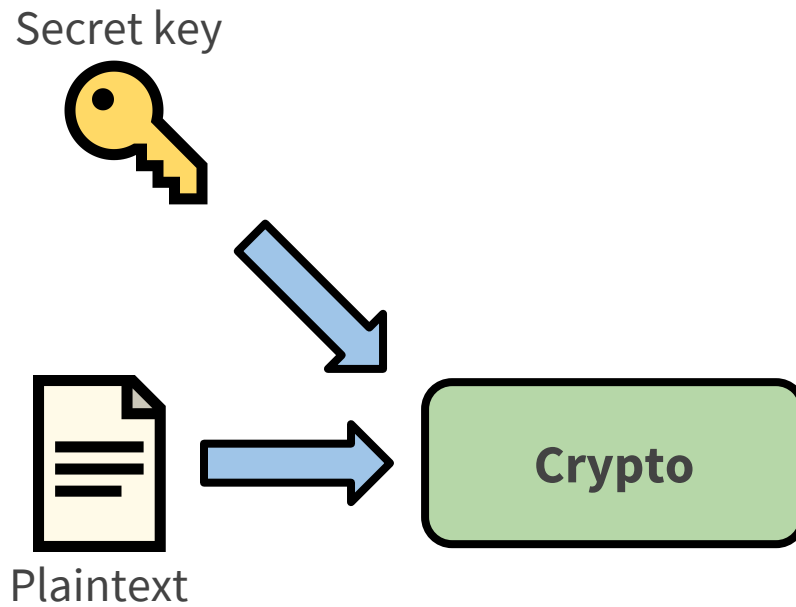
**Crypto**

# Timing side channels

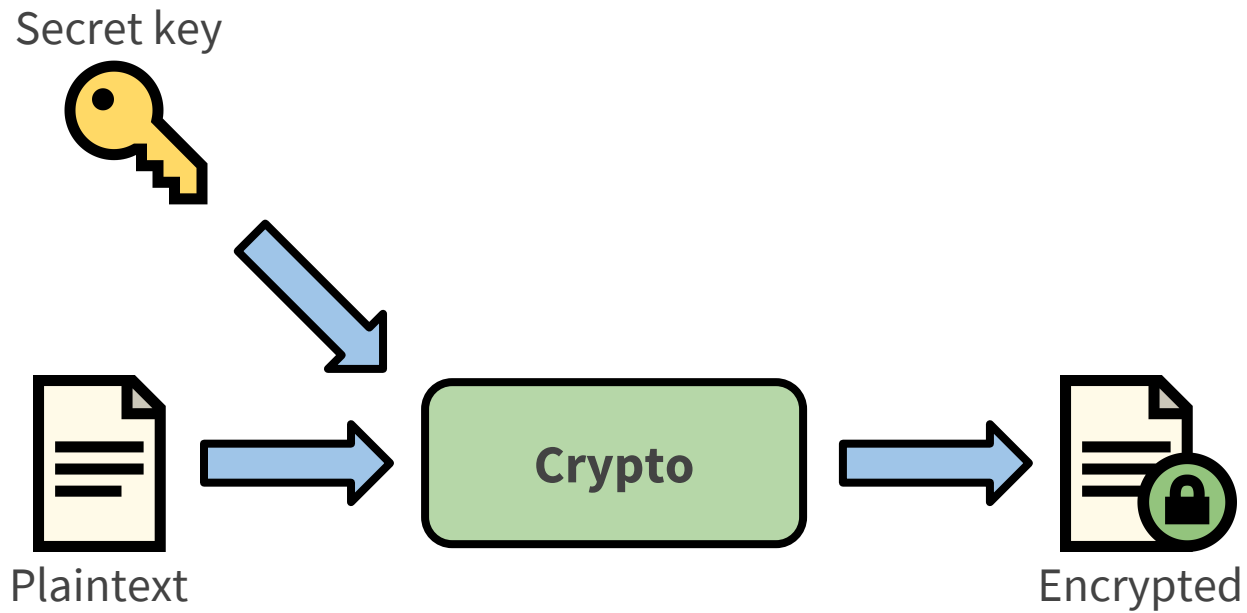
Secret key



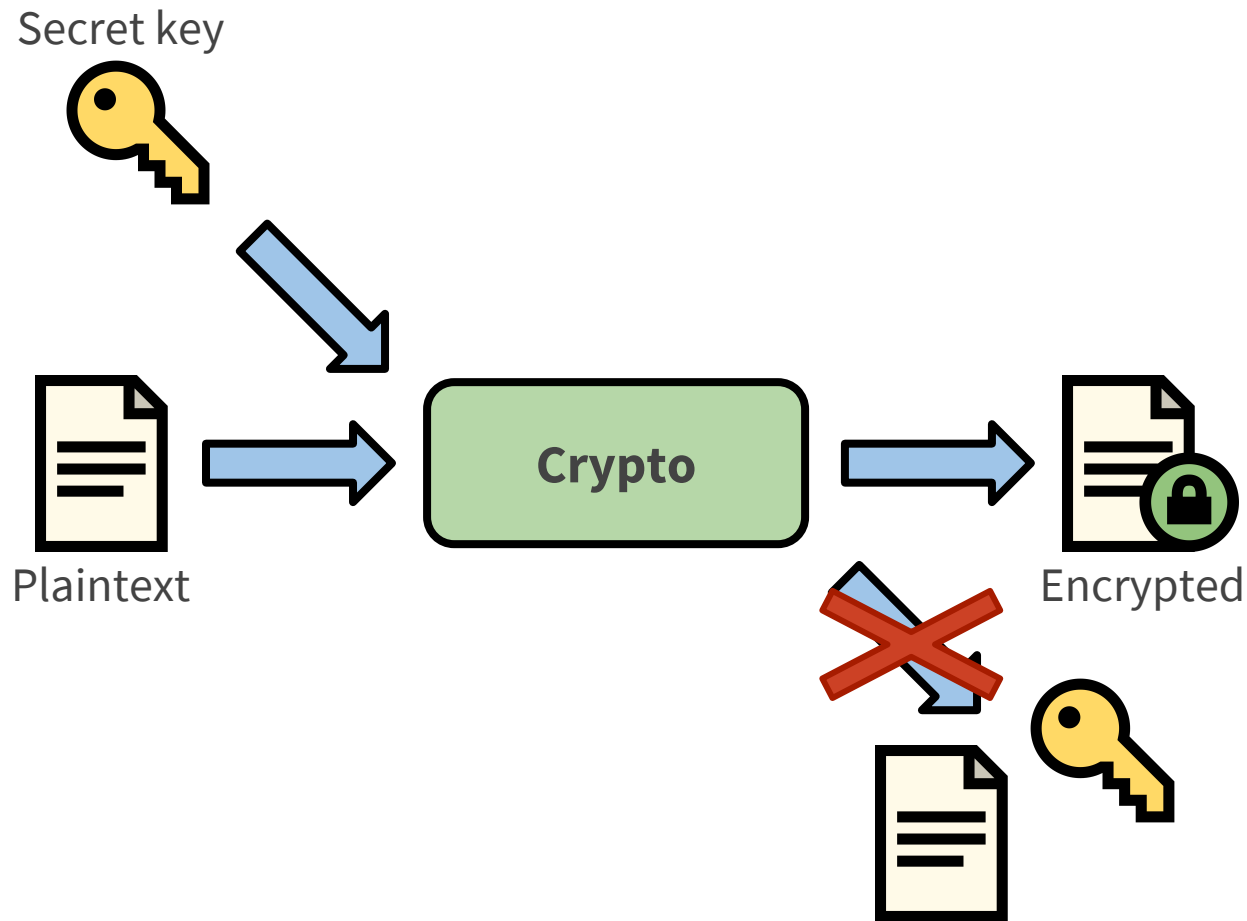
# Timing side channels



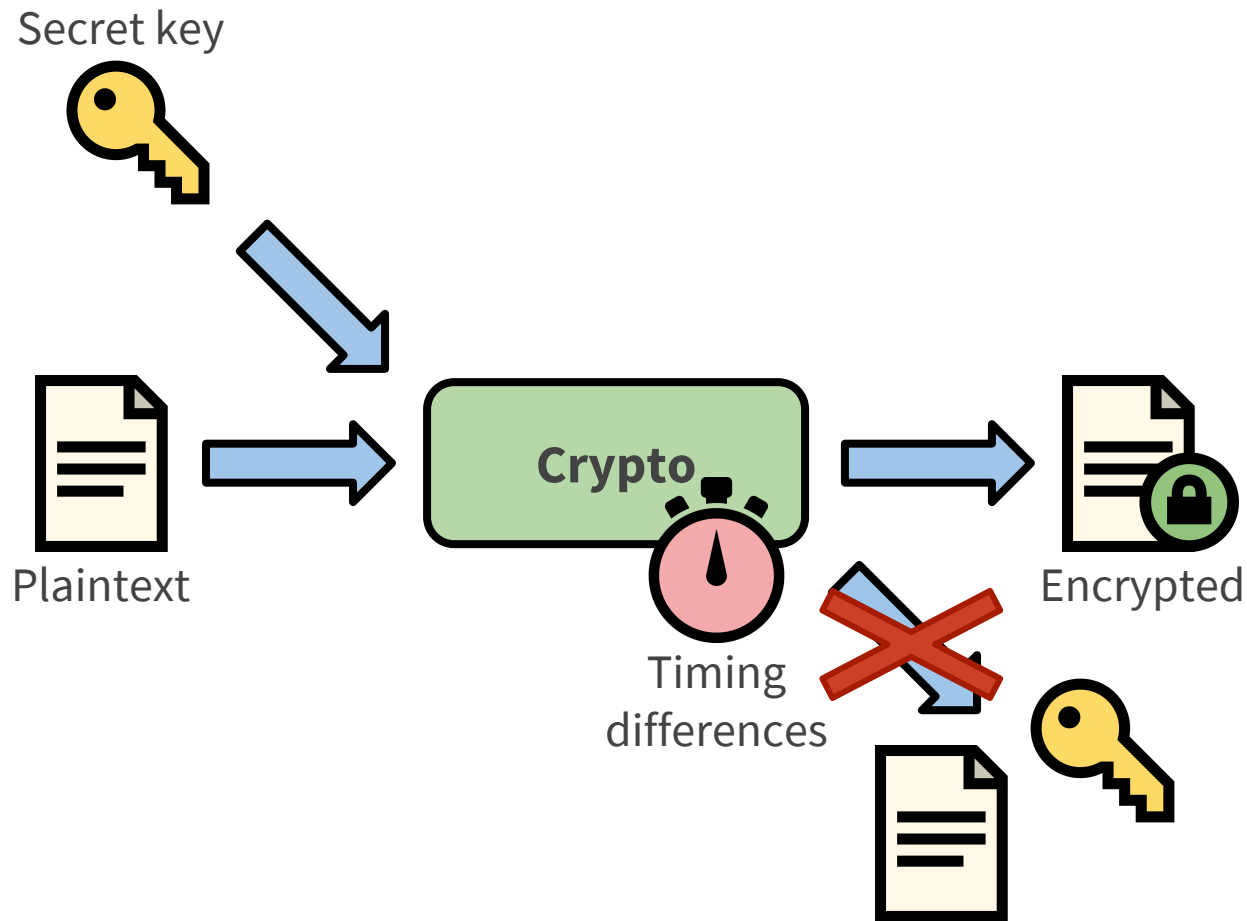
# Timing side channels



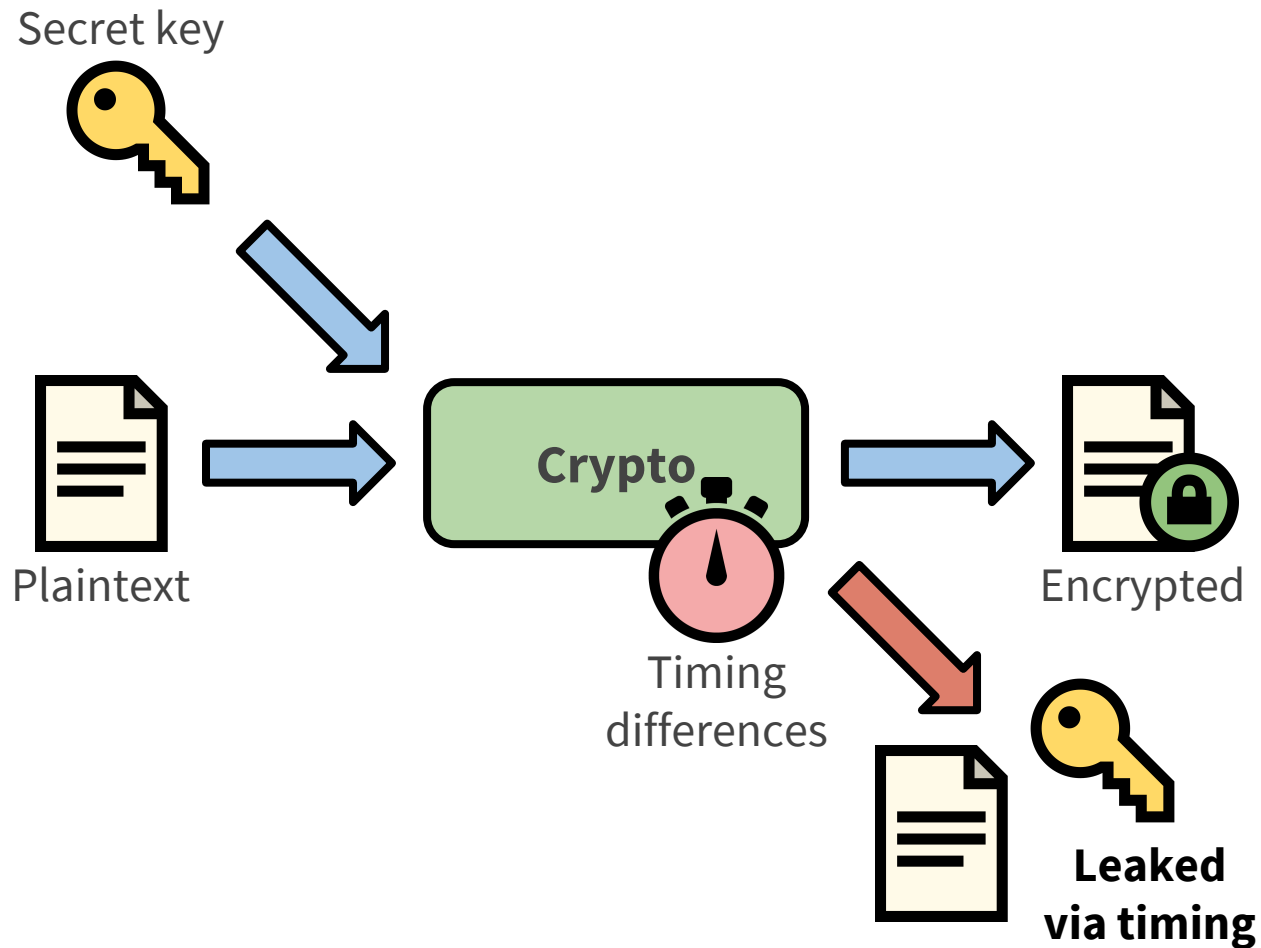
# Timing side channels



# Timing side channels

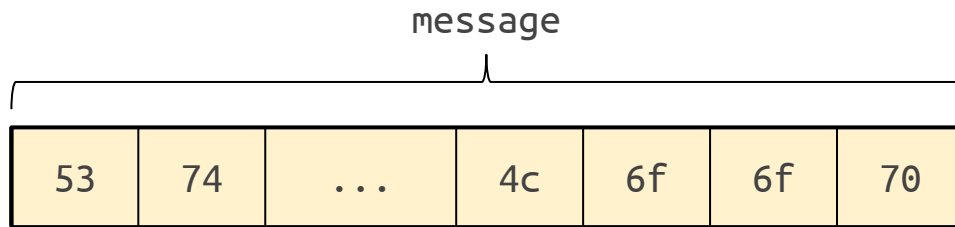


# Timing side channels

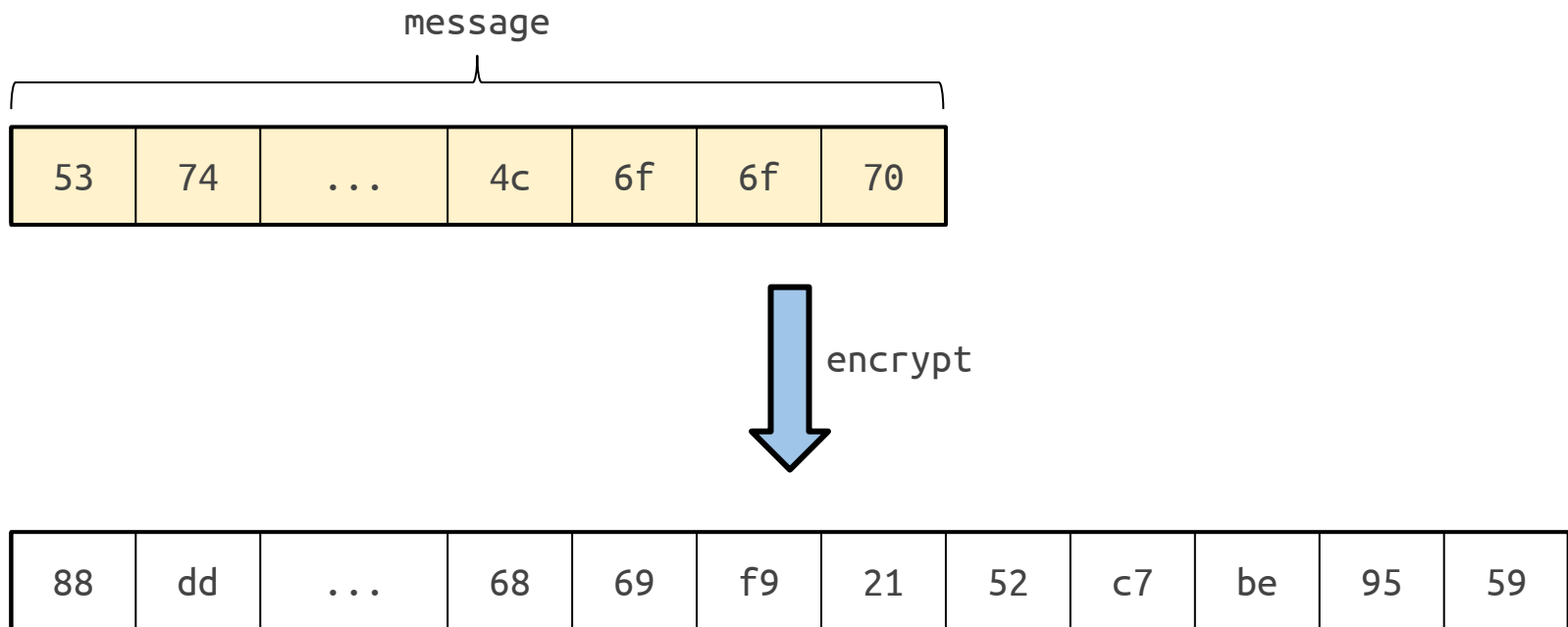




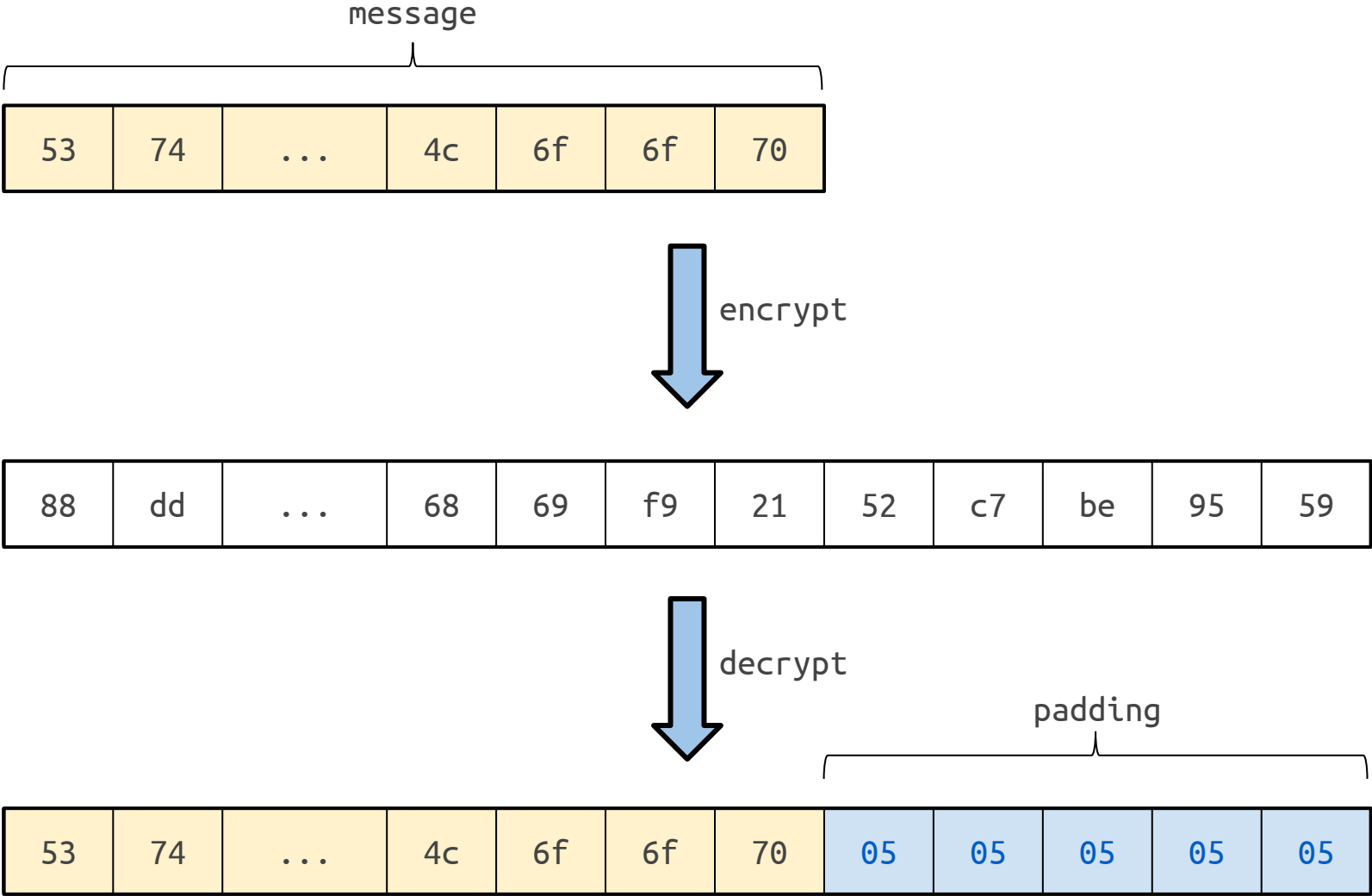
# Cryptographic padding



# Cryptographic padding

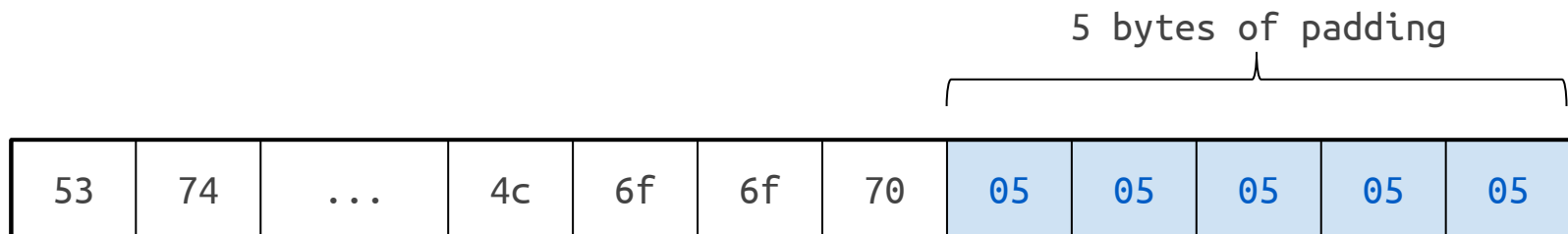


# Cryptographic padding



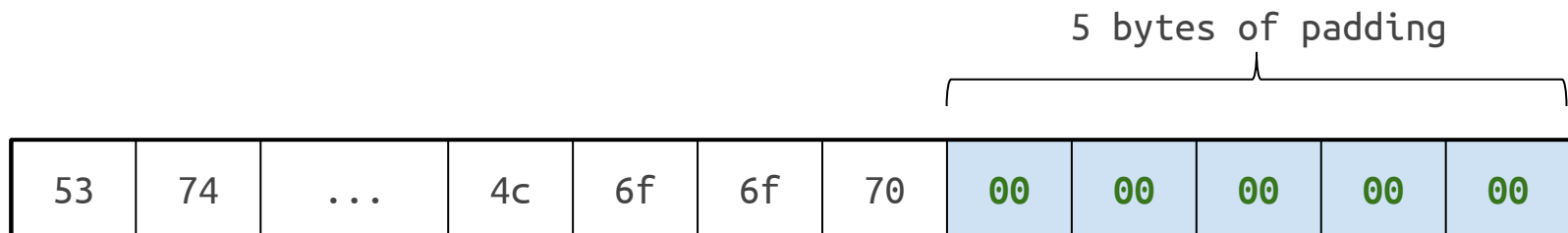
# Constant-time coding example

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



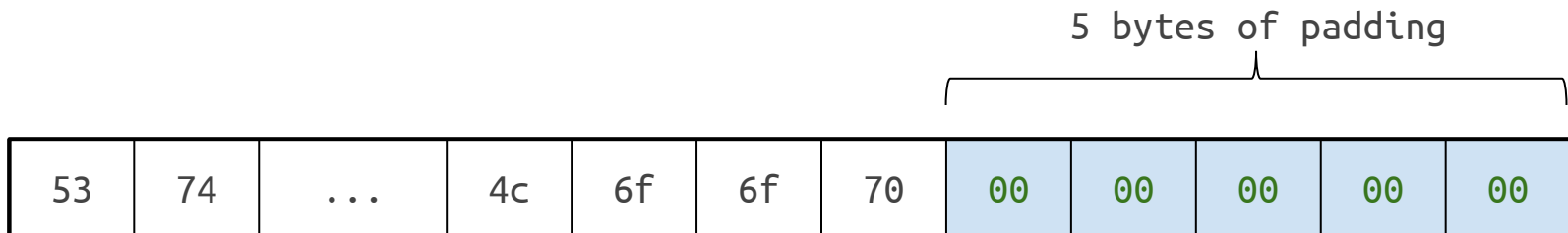
# Constant-time coding example

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



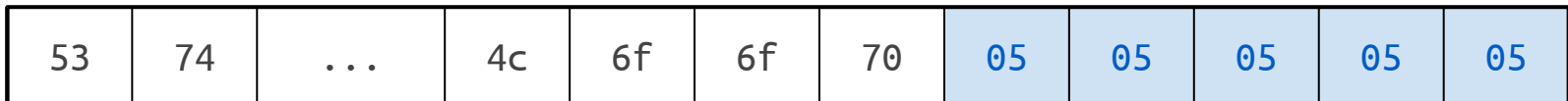
# Constant-time coding example

- 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!



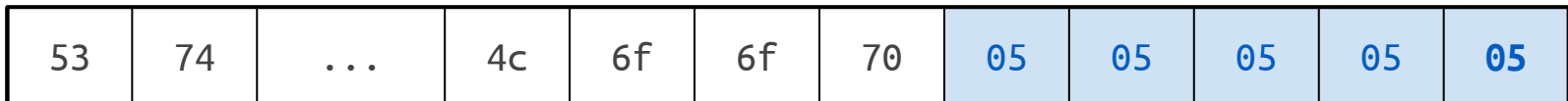
# Padding removal: 1st try

```
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;  
}
```



# Padding removal: 1st try

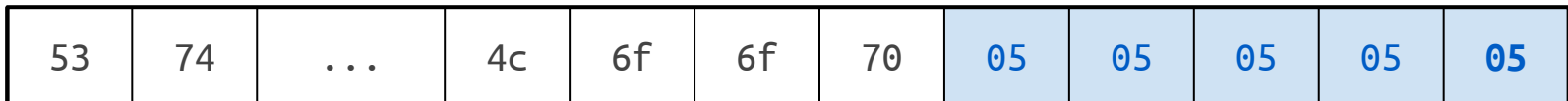
```
int32_t remove_padding(  
    uint8_t* buf,  
    uint32_t buflen) {  
  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i;  
    for (i = 0; i < padlen; i++) {  
        if (buf[buflen-i-1] != padlen)  
            return -1;  
        buf[buflen-i-1] = 0;  
    }  
    return padlen;  
}
```





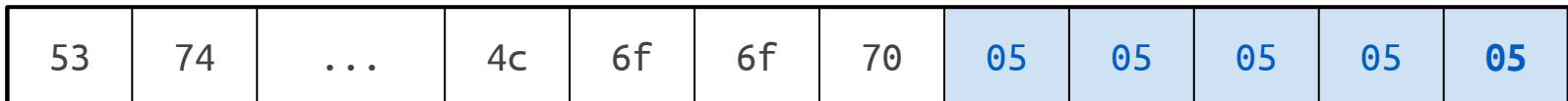
# Padding removal: 1st try

```
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;  
}
```



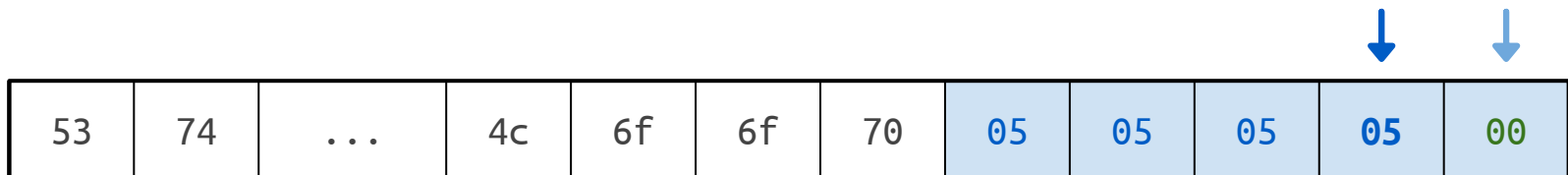
# Padding removal: 1st try

```
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;  
}
```



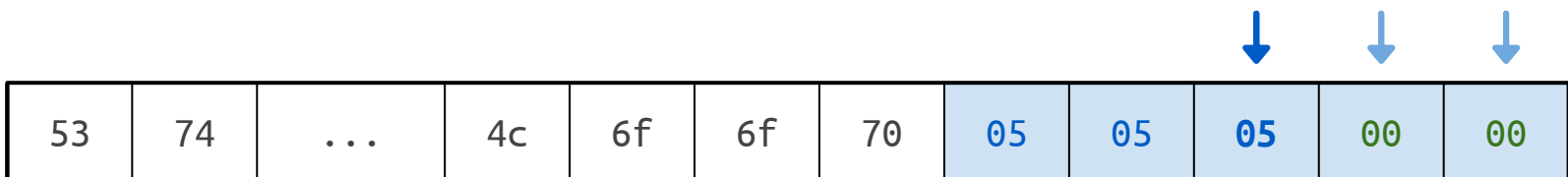
# Padding removal: 1st try

```
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;  
}
```



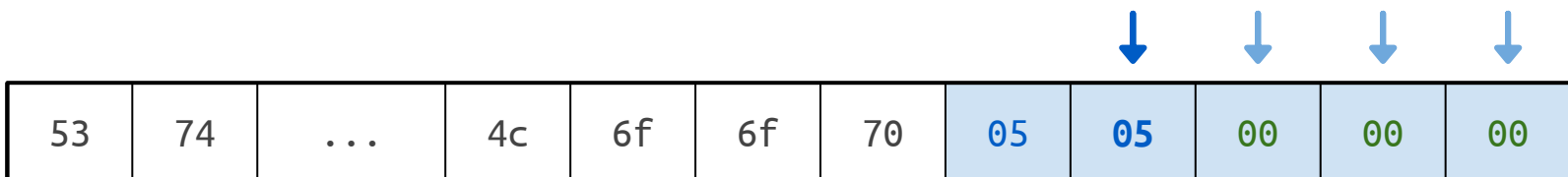
# Padding removal: 1st try

```
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;  
}
```



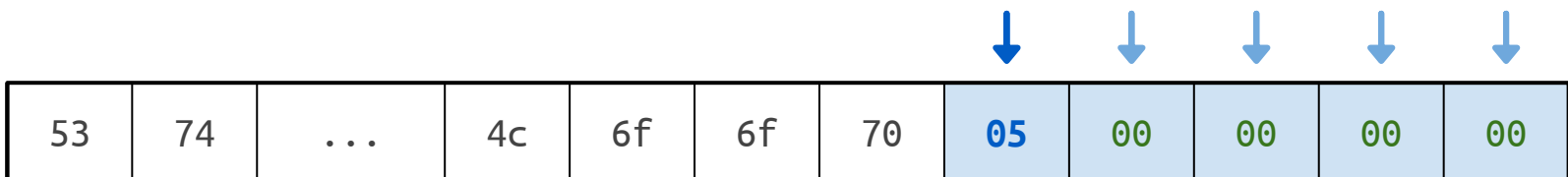
# Padding removal: 1st try

```
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;  
}
```



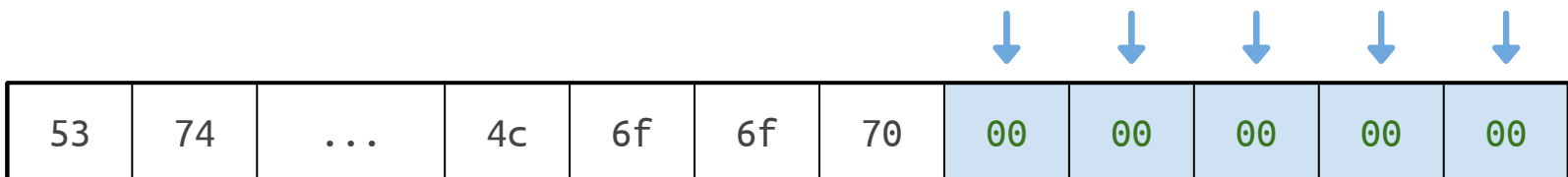
# Padding removal: 1st try

```
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;  
}
```



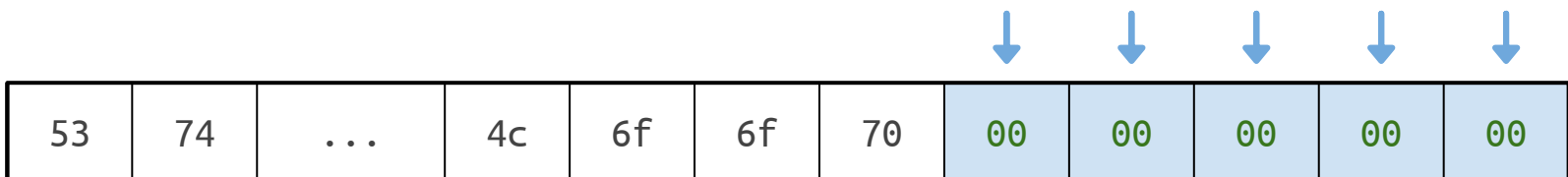
# Padding removal: 1st try

```
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;  
}
```



# Padding removal: 1st try

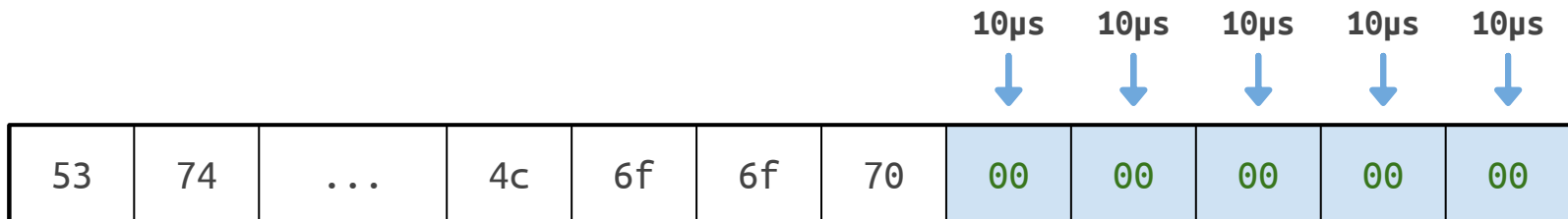
```
int32_t remove_padding(  
    uint8_t* buf,  
    uint32_t buflen) {  
  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i;  
    for (i = 0; i < padlen; i++) {  
        if (buf[buflen-i-1] != padlen)  
            return -1;  
        buf[buflen-i-1] = 0;  
    }  
    return padlen;  
}
```





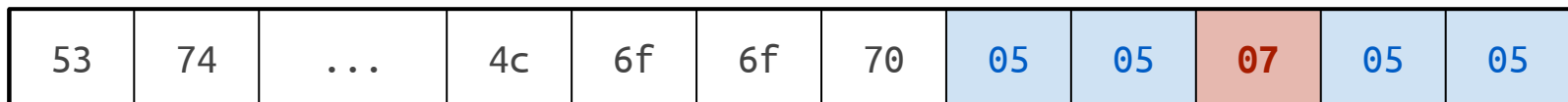
# Padding removal: 1st try

```
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;  
}
```



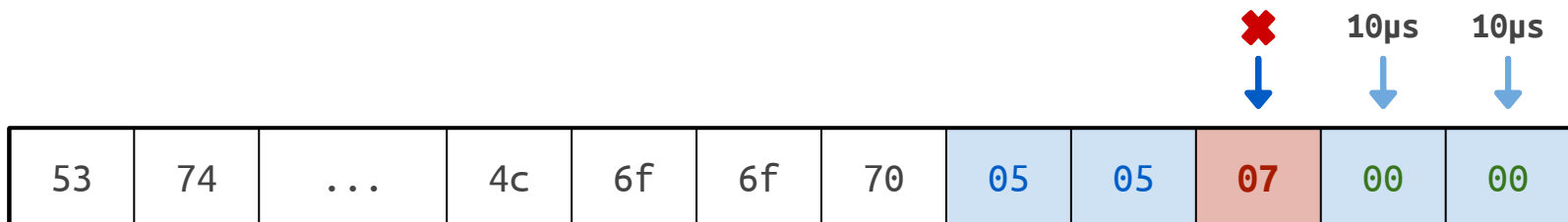
# Padding removal: 1st try

```
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;  
}
```



# Padding removal: 1st try

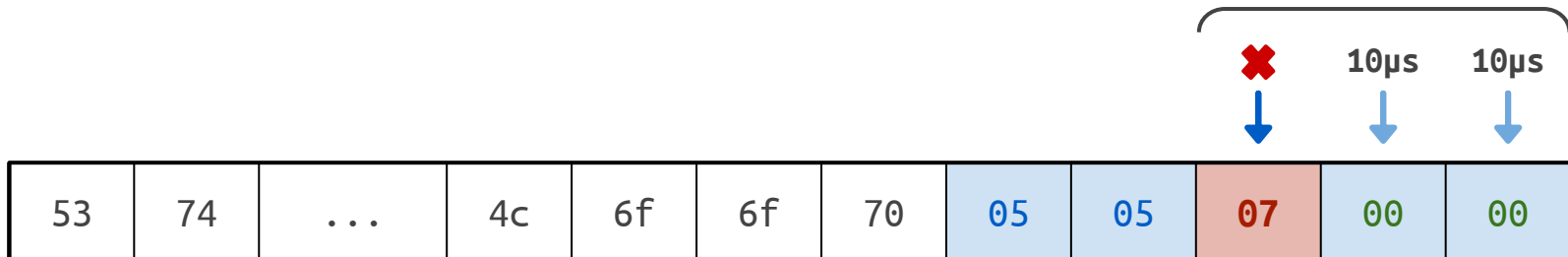
```
int32_t remove_padding(  
    uint8_t* buf,  
    uint32_t buflen) {  
  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i;  
    for (i = 0; i < padlen; i++) {  
        if (buf[buflen-i-1] != padlen)  
            return -1;  
        buf[buflen-i-1] = 0;  
    }  
    return padlen;  
}
```



# Padding removal: 1st try

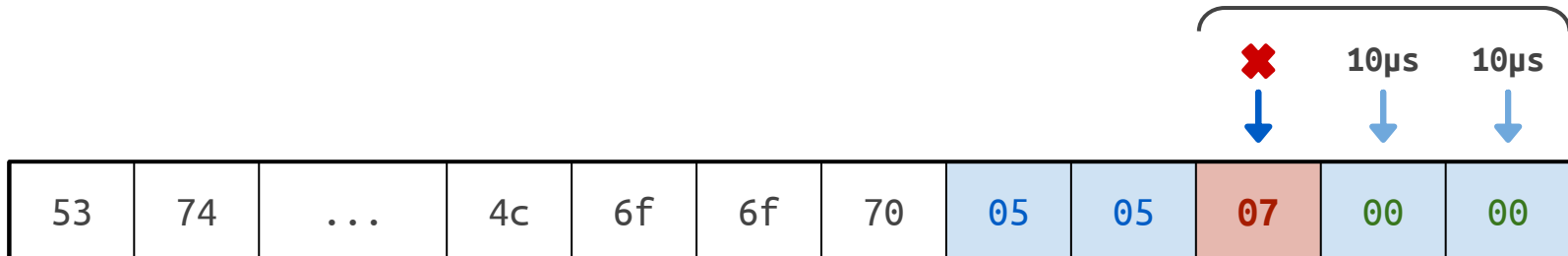
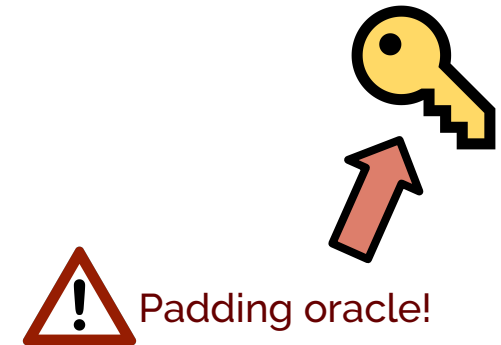
```
int32_t remove_padding(  
    uint8_t* buf,  
    uint32_t buflen) {  
  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i;  
    for (i = 0; i < padlen; i++) {  
        if (buf[buflen-i-1] != padlen)  
            return -1;  
        buf[buflen-i-1] = 0;  
    }  
    return padlen;  
}
```

 Padding oracle!



# Padding removal: 1st try

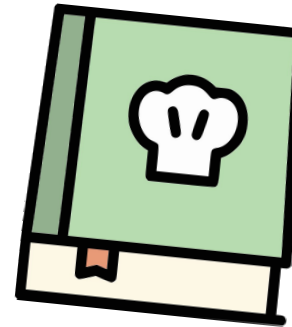
```
int32_t remove_padding(  
    uint8_t* buf,  
    uint32_t buflen) {  
  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i;  
    for (i = 0; i < padlen; i++) {  
        if (buf[buflen-i-1] != padlen)  
            return -1;  
        buf[buflen-i-1] = 0;  
    }  
    return padlen;  
}
```



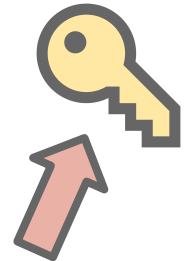
# Padding removal: 1st try

```
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.



Padding oracle!

✘

10µs

10µs

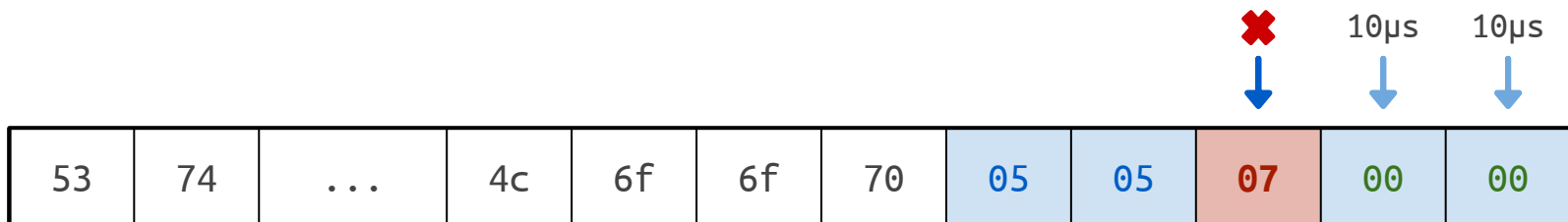


53	74	...	4c	6f	6f	70	05	05	07	00	00
----	----	-----	----	----	----	----	----	----	----	----	----

# Padding removal: 1st try

```
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;  
}
```

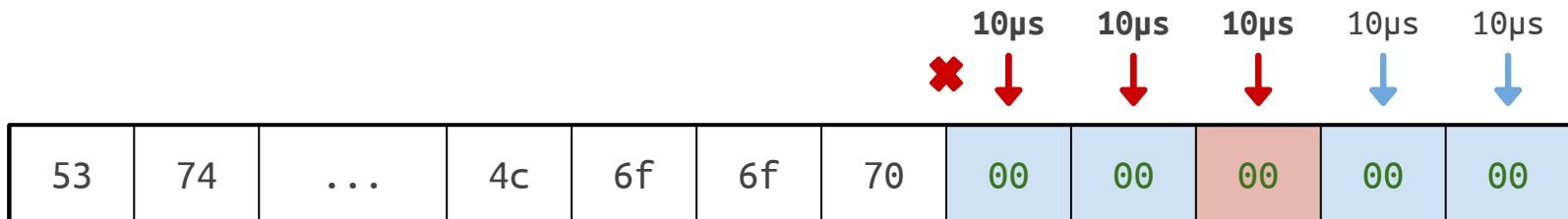
```
int32_t remove_padding2(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint8_t ok = 1;  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i;  
    for (i = 0; i < padlen; i++) {  
        if (buf[buflen-i-1] != padlen)  
            ok = 0;  
        buf[buflen-i-1] = 0;  
    }  
    return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



# Padding removal: 1st try

```
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;  
}
```

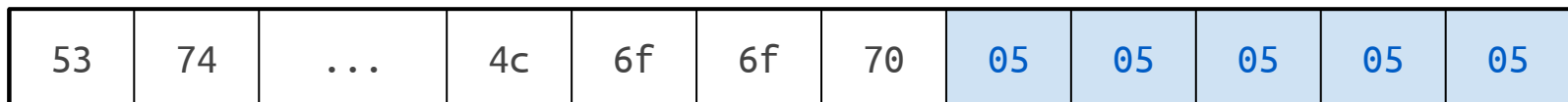
```
int32_t remove_padding2(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint8_t ok = 1;  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i;  
    for (i = 0; i < padlen; i++) {  
        if (buf[buflen-i-1] != padlen)  
            ok = 0;  
        buf[buflen-i-1] = 0;  
    }  
    return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





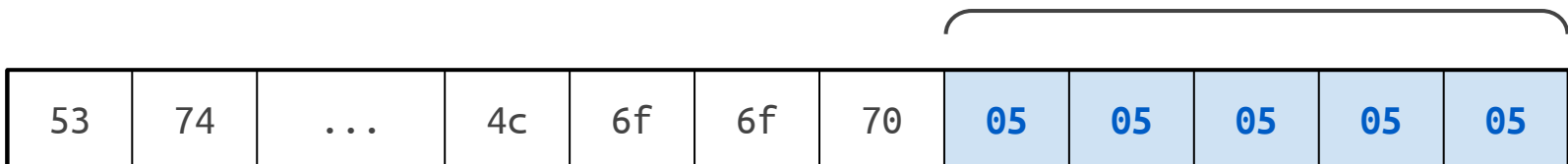
# Padding removal: 2nd try

```
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;  
}
```



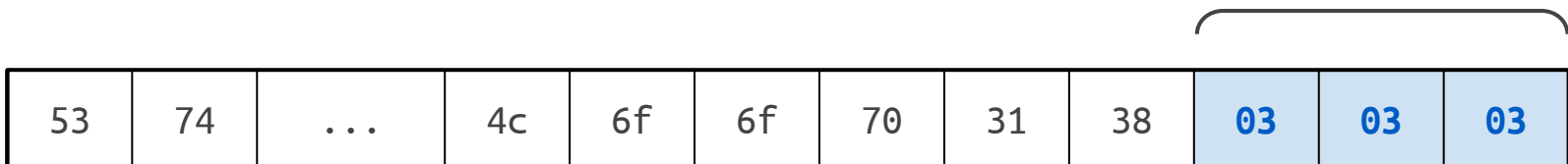
# Padding removal: 2nd try

```
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;  
}
```



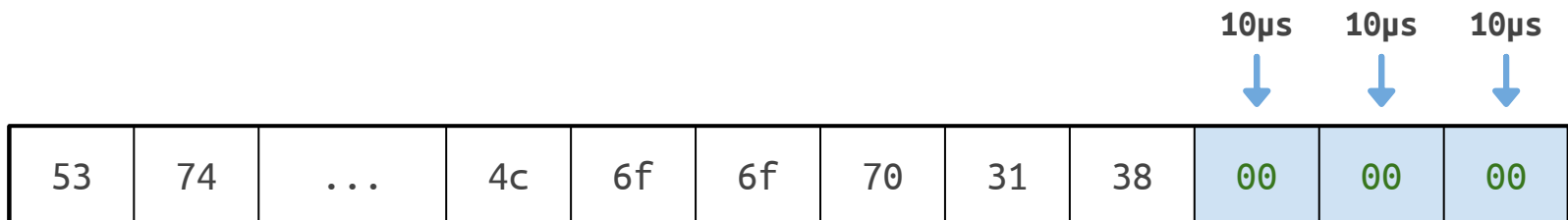
# Padding removal: 2nd try

```
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;  
}
```



# Padding removal: 2nd try

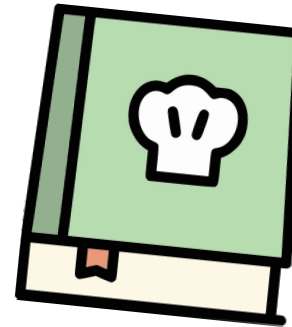
```
int32_t remove_padding2(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint8_t ok = 1;  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i;  
    for (i = 0; i < padlen; i++) {  
        if (buf[buflen-i-1] != padlen)  
            ok = 0;  
        buf[buflen-i-1] = 0;  
    }  
    return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



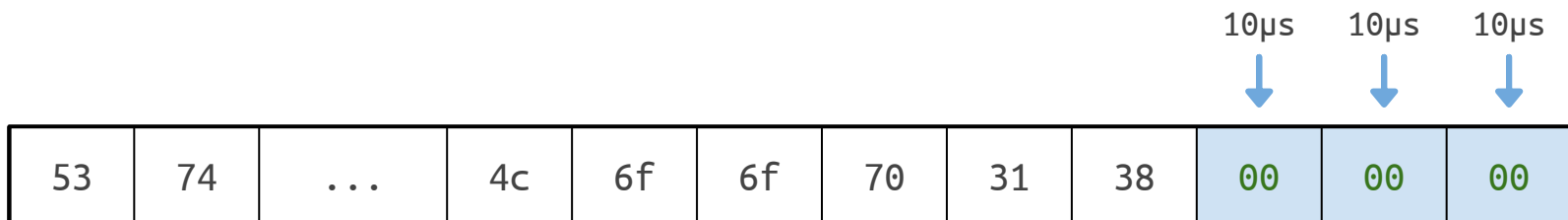
# Padding removal: 2nd try

```
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!



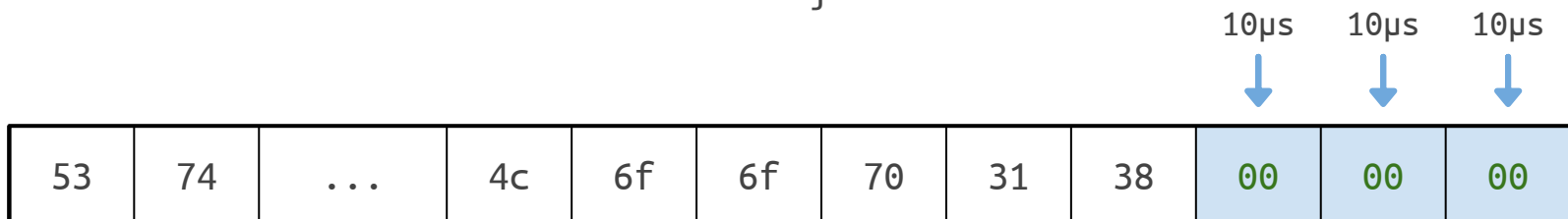
Use this instead.



# Padding removal: 2nd try

```
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;  
}
```

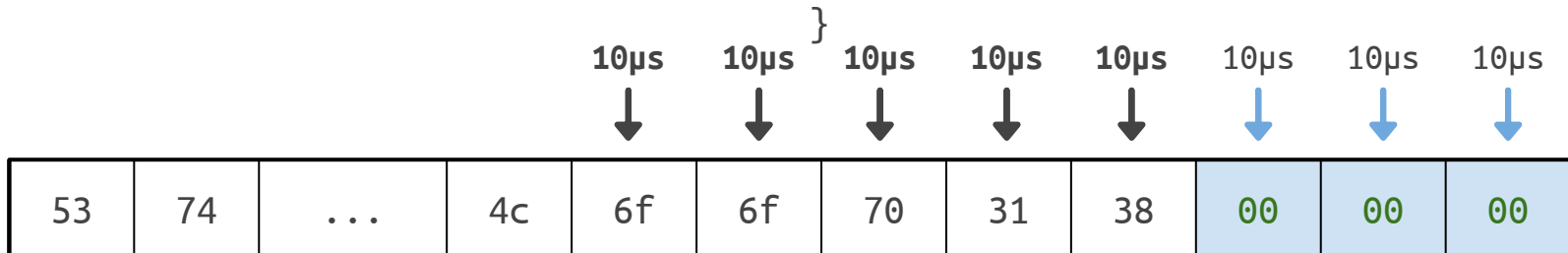
```
int32_t remove_padding3(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint8_t ok = 1;  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i;  
    for (i = buflen-255; i < buflen; i++) {  
        uint8_t b = buf[i];  
        if (i >= buflen - padlen) {  
            if (b != padlen)  
 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



# Padding removal: 2nd try

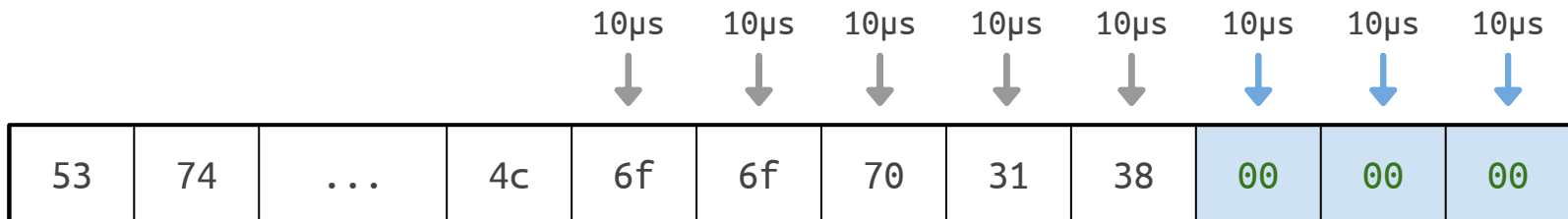
```
int32_t remove_padding2(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint8_t ok = 1;  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i;  
    for (i = 0; i < padlen; i++) {  
        if (buf[buflen-i-1] != padlen)  
            ok = 0;  
        buf[buflen-i-1] = 0;  
    }  
    return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



# Padding removal: 3rd try

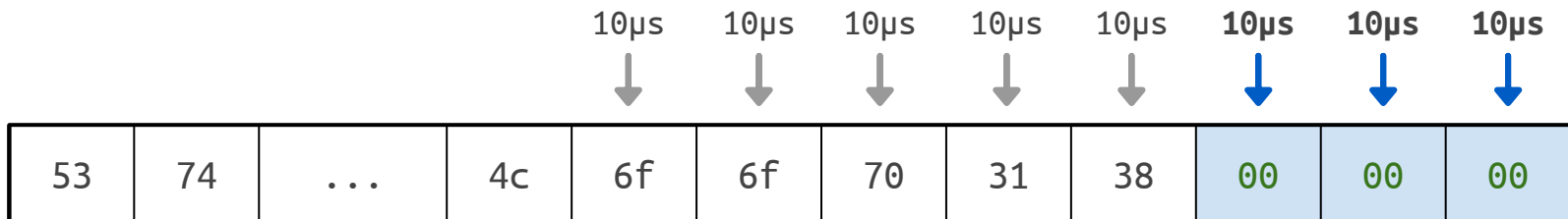
```
int32_t remove_padding3(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint8_t ok = 1;  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i;  
    for (i = buflen-255; i < buflen; i++) {  
        uint8_t b = buf[i];  
        if (i >= buflen - padlen) {  
            if (b != padlen)  
 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





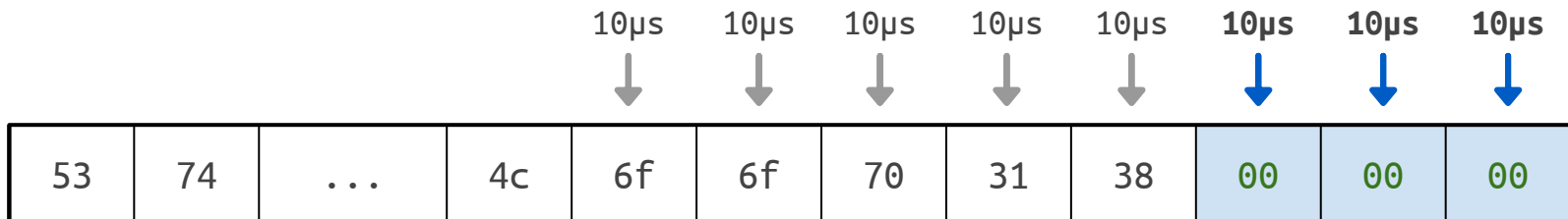
# Padding removal: 3rd try

```
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;  
}
```



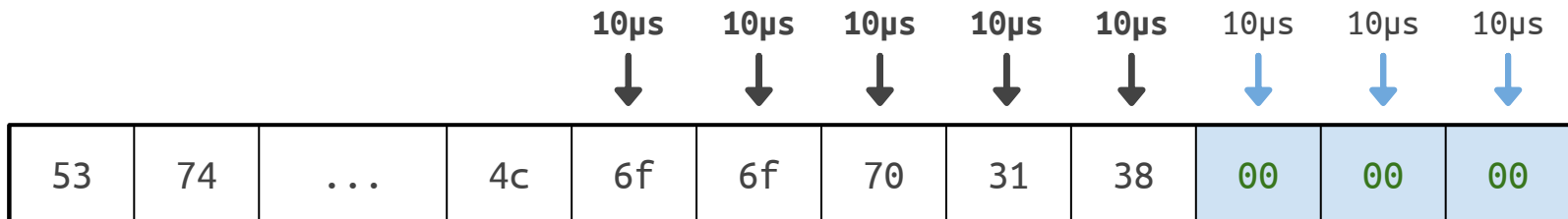
# Padding removal: 3rd try

```
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;  
}
```



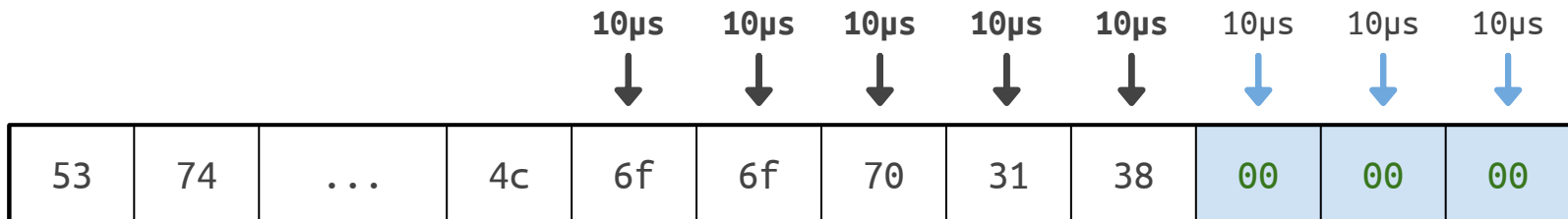
# Padding removal: 3rd try

```
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;  
}
```



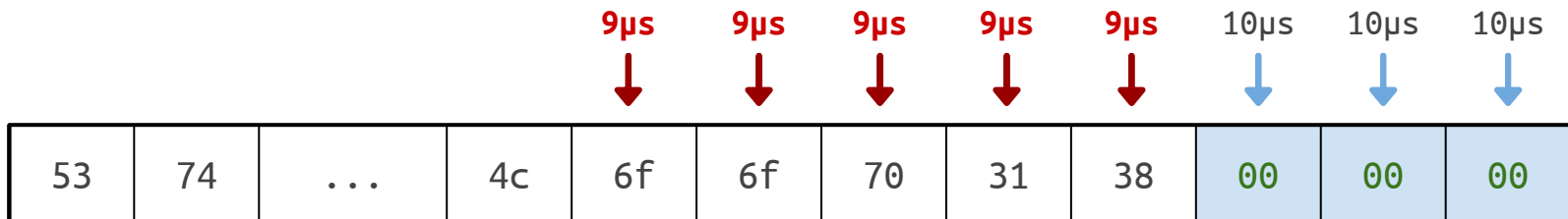
# Padding removal: 3rd try

```
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;  
}
```



# Padding removal: 3rd try

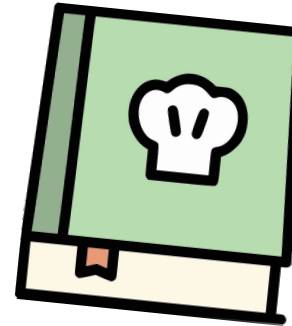
```
int32_t remove_padding3(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint8_t ok = 1;  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i;  
    for (i = buflen-255; i < buflen; i++) {  
        uint8_t b = buf[i];  
        if (i >= buflen - padlen) {  
            if (b != padlen)  
 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



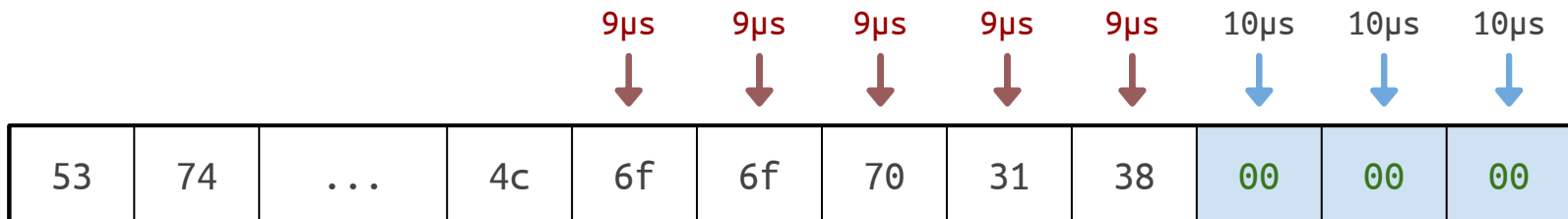
# Padding removal: 3rd try

```
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!



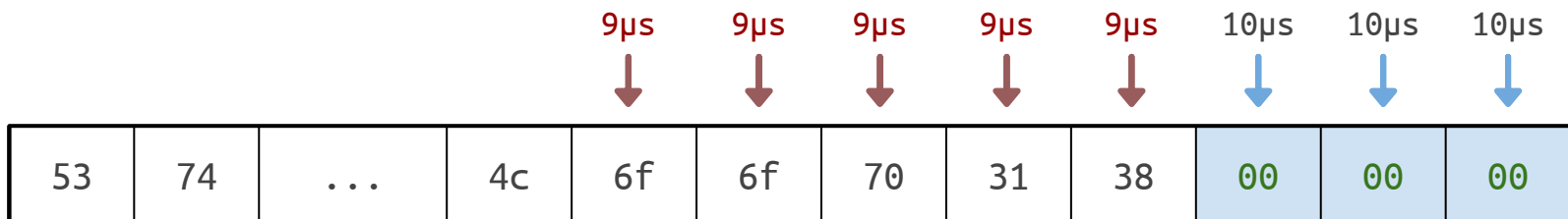
Use this instead.



# Padding removal: 3rd try

```
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_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 improper_index =  
            -(i - (buflen - padlen) >> 31);  
        uint32_t matches_pad =  
            -((b ^ padlen) - 1 >> 31);  
        ok &= matches_pad | improper_index;  
        b = improper_index & b;  
        buf[i] = b;  
    }  
    return (ok & padlen) | ~ok;  
}
```

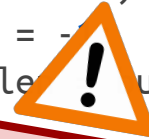


# Padding removal: 3rd try

```
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_padding4(
    uint8_t* buf,
    uint32_t buflen) {
    uint32_t ok = 1;
    uint8_t padlen = buf[buflen-1];
    for (uint32_t i = buflen-255; i < buflen; i++) {
        uint8_t b = buf[i];
        uint32_t improper_index =
            -(i - (buflen - padlen) >> 31);
        uint32_t matches_pad =
            -((b ^ padlen) - 1 >> 31);
        ok &= matches_pad | improper_index;
        b = improper_index & b;
        buf[i] = b;
    }
    return (ok & padlen) | ~ok;
}
```

**Ugly! Do not read!**



9µs    9µs    9µs    9µs    9µs    10µs    10µs    10µs

↓       ↓       ↓       ↓       ↓       ↓       ↓       ↓

53	74	...	4c	6f	6f	70	31	38	00	00	00
----	----	-----	----	----	----	----	----	----	----	----	----

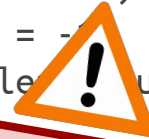


# Padding removal: 3rd try

```
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_padding4(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint32_t ok = 1;  
    uint8_t padlen = buf[buflen-1];  
    for (uint32_t i = buflen-255; i < buflen; i++) {  
        uint8_t b = buf[i];  
        uint32_t improper_index =  
            -(i - (buflen - padlen) >> 31);  
        uint32_t matches_pad =  
            -((b ^ padlen) - 1 >> 31);  
        ok &= matches_pad | improper_index;  
        b = improper_index & b;  
        buf[i] = b;  
    }  
    return (ok & padlen) | ~ok;  
}
```

**Ugly! Do not read!**



12µs   12µs   12µs   12µs   12µs   12µs   12µs   12µs

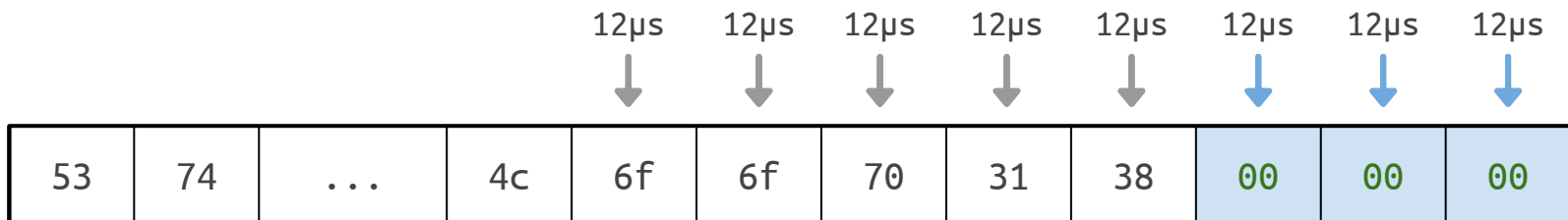
↓   ↓   ↓   ↓   ↓   ↓   ↓   ↓

53	74	...	4c	6f	6f	70	31	38	00	00	00
----	----	-----	----	----	----	----	----	----	----	----	----

# Padding removal: 4th try

```
int32_t remove_padding4(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint32_t ok = 0;  
    uint8_t padlen = 0;  
    uint32_t i = buflen - 1;  
    for (i = buflen - 255; i < buflen; i++) {  
        uint8_t b = buf[i];  
        uint32_t improper_index = -(i - (buflen - padlen) >> 31);  
        uint32_t matches_pad = -((b ^ padlen) - 1 >> 31);  
        ok &= matches_pad | improper_index;  
        b = improper_index & b;  
        buf[i] = b;  
    }  
    return (ok & padlen) | ~ok;  
}
```

Ugly! Do not read!

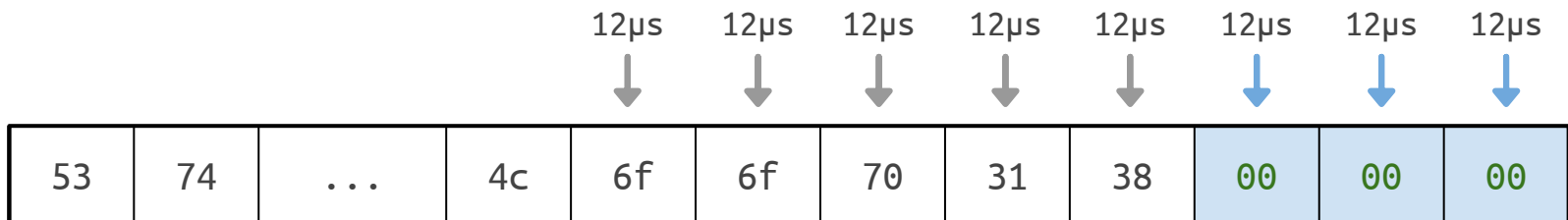


# Padding removal: 4th try

```
int32_t remove_padding4(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint32_t ok = 0;  
    uint8_t padlen = 0;  
    uint32_t i = buflen - 1;  
    for (i = buflen - 255; i < buflen; i++) {  
        uint8_t b = buf[i];  
        uint32_t improper_index = -(i - (buflen - padlen) >> 31);  
        uint32_t matches_pad = -((b ^ padlen) - 1 >> 31);  
        ok &= matches_pad | improper_index;  
        b = improper_index & b;  
        buf[i] = b;  
    }  
    return (ok & padlen) | ~ok;  
}
```

**Ugly! Do not read!**

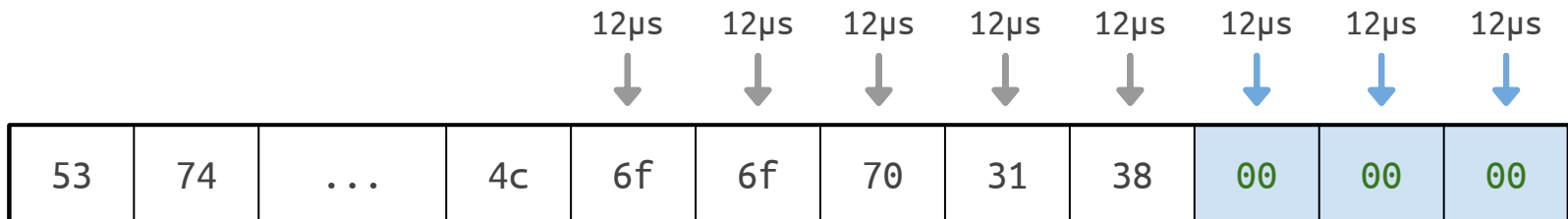
???



# Padding removal: 4th try

```
int32_t remove_padding4(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint32_t ok = 0;  
    uint8_t padlen = buf[buflen-1];  
    uint32_t i = 0;  
    for (i = buflen-255; i < buflen; i++) {  
        uint8_t b = buf[i];  
        uint32_t improper_index = -(i - (buflen - padlen) >> 31);  
        uint32_t matches_pad = -((b ^ padlen) - 1 >> 31);  
        ok &= matches_pad | improper_index;  
        b = improper_index & b;  
        buf[i] = b;  
    }  
    return (ok & padlen) | ~ok;  
}
```

**Ugly! Do not read!**

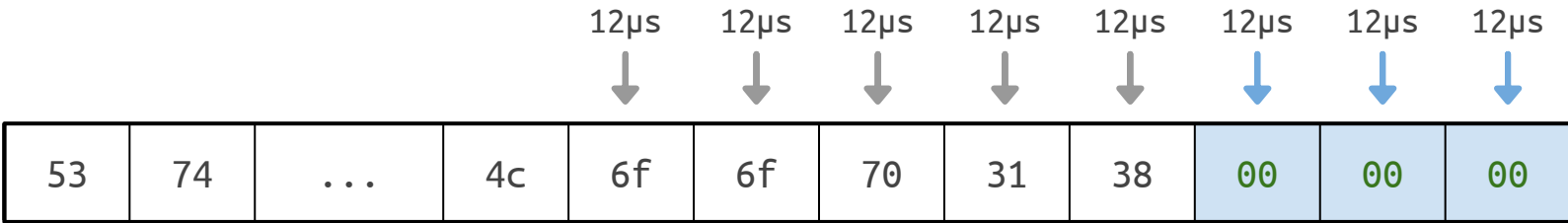


# Padding removal: 4th try

```
int32_t remove_padding4(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint32_t ok = 0;  
    uint8_t padlen = 0;  
    uint32_t i = 0;  
    for (i = buflen - 255; i < buflen; i++) {  
        uint8_t b = buf[i];  
        uint32_t improper_index = -(i - (buflen - padlen) >> 31);  
        uint32_t matches_pad = -((b ^ padlen) - 1 >> 31);  
        ok &= matches_pad | improper_index;  
        b = improper_index & b;  
        buf[i] = b;  
    }  
    return (ok & padlen) | ~ok;  
}
```

**Ugly! Do not read!**

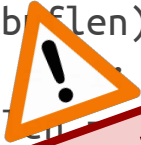
} random\_sleep();



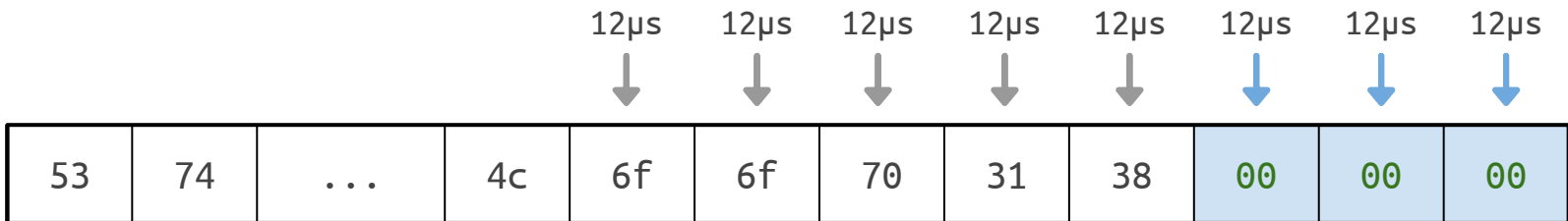
# Padding removal: 4th try

```
int32_t remove_padding4(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint32_t ok = 0;  
    uint8_t padlen = 0;  
    uint32_t i = 0;  
    for (i = buflen-255; i < buflen; i++) {  
        uint8_t b = buf[i];  
        uint32_t improper_index = -(i - (buflen - padlen) >> 31);  
        uint32_t matches_pad = -((b ^ padlen) - 1 >> 31);  
        ok &= matches_pad | improper_index;  
        b = improper_index & b;  
        buf[i] = b;  
    }  
    return (ok & padlen) | ~ok;  
}
```

**Ugly! Do not read!**



} random\_sleep();

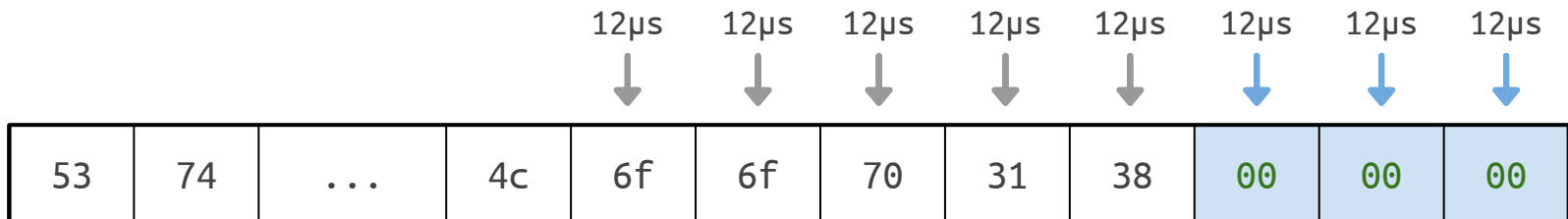


# Padding removal: 4th try

```
int32_t remove_padding4(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint32_t ok = 0;  
    uint8_t padlen = 0;  
    uint32_t i = 0;  
    for (i = buflen - 255; i < buflen; i++) {  
        uint8_t b = buf[i];  
        uint32_t improper_index = -(i - (buflen - padlen) >> 31);  
        uint32_t matches_pad = -((b ^ padlen) - 1 >> 31);  
        ok &= matches_pad | improper_index;  
        b = improper_index & b;  
        buf[i] = b;  
    }  
    return (ok & padlen) | ~ok;  
}
```

**Ugly! Do not read!**

} sleep\_til\_max();

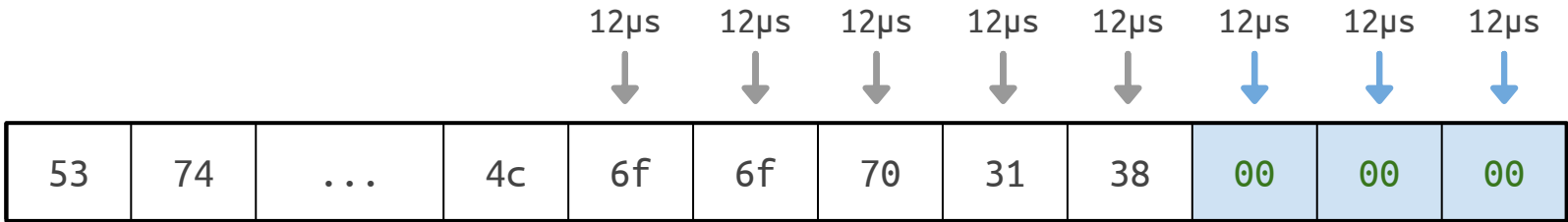


# Padding removal: 4th try

```
int32_t remove_padding4(  
    uint8_t* buf,  
    uint32_t buflen) {  
    uint32_t ok = 0;  
    uint8_t padlen = 0;  
    uint32_t i = buflen - 1;  
    for (i = buflen - 255; i < buflen; i++) {  
        uint8_t b = buf[i];  
        uint32_t improper_index = -(i - (buflen - padlen) >> 31);  
        uint32_t matches_pad = -((b ^ padlen) - 1 >> 31);  
        ok &= matches_pad | improper_index;  
        b = improper_index & b;  
        buf[i] = b;  
    }  
    return (ok & padlen) | ~ok;  
}
```

**Ugly! Do not read!**

**X**  
sleep\_til\_max();





# Writing constant-time code is hard

- **Challenge:** manually ensuring code is CT
  - Manually keep track of secret vs. public
  - Standard programming constructs introduce timing leaks
  - Can't simply pad/randomize

# Writing constant-time code is hard

- **Challenge:** manually ensuring code is CT
  - Manually keep track of secret vs. public
  - Standard programming constructs introduce timing leaks
  - Can't simply pad/randomize
  
- **Consequence:** vulnerabilities!
  - Difficult to write correct code
  - Hard to understand what CT code is doing
  - Hard to maintain CT code

# Error-prone in practice

```
384 384     SSL_RECORD *rr;
385 385     unsigned int mac_size;
386 386     unsigned char md[EVP_MAX_MD_SIZE];
387 387     int decryption_failed_or_bad_record_mac = 0;
388 387
389 388
390 389     rrr = &(s->s3->rrec);
391 389
392 390     /* -417,13 +418,10 @@ dtls1_process_record(SSL *s)
393 390
394 391     enc_err = s->method->ssl3_enc->enc(s,0);
395 391     if (enc_err <= 0)
396 391     {
397 391         /* decryption failed, silently discard message */
398 391         if (enc_err < 0)
399 391         {
400 391             {
401 391                 rrr->length = 0;
402 391                 s->packet_length = 0;
403 391             }
404 391             goto err;
405 391         }
406 391         /* To minimize information leaked via timing, we will always
407 391          * perform all computations before discarding the message.
408 391          */
409 391         decryption_failed_or_bad_record_mac = 1;
410 391     }
411 391
412 391     #ifdef TLS_DEBUG
413 391     /* -453,7 +451,7 @@ printf("\n");
414 391
415 391     SSLerr(SSL_F_DTLS1_PROCESS_RECORD,SSL_R_PRE_MAC_LENGTH_TOO_LONG);
416 391     goto f_err;
417 391
418 391     #else
419 391     goto err;
420 391     #endif
421 391     decryption_failed_or_bad_record_mac = 1;
422 391
423 391     #endif
424 391     }
425 391     /* check the MAC for rrr->input (it's in mac_size bytes at the tail) */
426 391     /* -464,17 +462,25 @@ printf("\n");
427 391
428 391     SSLerr(SSL_F_DTLS1_PROCESS_RECORD,SSL_R_LENGTH_TOO_SHORT);
429 391     goto f_err;
430 391
431 391     #else
432 391     goto err;
433 391     #endif
434 391     decryption_failed_or_bad_record_mac = 1;
435 391
436 391     #endif
437 391     }
438 391     rrr->length=mac_size;
439 391     !s->method->ssl3_enc->mac(s,md,0);
440 391     if (1 < 0 || memcmp(md,&(rr->data[rr->length]),mac_size) != 0)
441 391     {
442 391         goto err;
443 391         decryption_failed_or_bad_record_mac = 1;
444 391     }
445 391     }
446 391
447 391
448 391     if (decryption_failed_or_bad_record_mac)
449 391     {
450 391         /* decryption failed, silently discard message */
451 391         rrr->length = 0;
452 391         s->packet_length = 0;
453 391         goto err;
454 391     }
455 391
456 391     /* r->length is now just compressed */
457 391     if (s->expand != NULL)
458 391     {
```

## OpenSSL padding oracle attack

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

# Error-prone in practice

```
384 384     SSL3_RECORD *rr;
385 385     unsigned int mac_size;
386 386     unsigned char
387 387     int decryption;
388 387     rrr = &(s->s3->
389 388     /* de
390 389     if (e
417 418     enc_err = s->
419 419     if (enc_err <
420 420     {
421 421     /* de
422 422     if (e
423 423     +
424 424     +
425 425     +
426 426     goto
427 427     /* To
428 428     +
429 429     #ifdef TLS_DEBUG
430 430     /* -453,7 +451,7 00 p
453 451     +
454 452     +
455 453     #else
456 454     +
457 455     #endif
458 456     /* ch
459 457     /* -464,17 +462,25 00
464 462     +
465 463     +
466 464     #else
467 465     +
468 466     #endif
469 468     rrr->l
470 469     lns->
471 469     if (1
472 470     +
473 471     +
474 472     +
475 473     }
476 474     }
477 475     }
478 476     if (decryption
479 477     {
480 478     /* de
481 479     rrr->l
482 480     s->npad
483 481     goto
484 482     }
485 483     }
486 484     /* r->length
487 485     if (s->expand
488 486     {
755 -     EVP_DigestUpdate(&md_ctx,md,2);
756 -     EVP_DigestUpdate(&md_ctx,rec->input,rec->length);
757 -     EVP_DigestFinal_ex( &md_ctx,md,NULL);
758 -
759 -     EVP_MD_CTX_copy_ex( &md_ctx,hash);
760 -     EVP_DigestUpdate(&md_ctx,mac_sec,md_size);
761 -     EVP_DigestUpdate(&md_ctx,ssl3_pad_2,npad);
762 -     EVP_DigestUpdate(&md_ctx,md,md_size);
763 -     EVP_DigestFinal_ex( &md_ctx,md,&md_size);
764 -
765 -     EVP_MD_CTX_cleanup(&md_ctx);
734 +     if (!send &&
735 +         EVP_CIPHER_CTX_mode(ssl->enc_read_ctx) == EVP_CIPHER_CBC_MODE &&
736 +         ssl3_cbc_record_digest_supported(hash))
737 +     {
738 +         /* This is a CBC-encrypted record. We must avoid leaking any
739 +          * timing-side channel information about how many blocks of
740 +          * data we are hashing because that gives an attacker a
741 +          * timing-oracle. */
742 +
743 +         /* npad is, at most, 48 bytes and that's with MD5:
744 +          * 16 + 48 + 8 (sequence bytes) + 1 + 2 = 75.
745 +          *
746 +          * With SHA-1 (the largest hash speeded for SSLv3) the hash size
747 +          * goes up 4, but npad goes down by 8, resulting in a smaller
748 +          * total size. */
749 +         unsigned char header[75];
750 +         unsigned j = 0;
751 +         memcpy(header+j, mac_sec, md_size);
752 +         j += md_size;
753 +         memcpy(header+j, ssl3_pad_1, npad);
754 +         j += npad;
755 +         memcpy(header+j, seq, 8);
756 +         j += 8;
757 +         header[j++] = rec->type;
758 +         header[j++] = rec->length >> 8;
759 +         header[j++] = rec->length & 0xff;
760 +
761 +         ssl3_cbc_digest_record(
762 +             hash,
763 +             md, &md_size,
764 +             header, rec->input,
765 +             rec->length + md_size, rec->orig_len,
766 +             mac_sec, md_size,
767 +             1 /* is SSLv3 */);
768 +     }
769 +     else
770 +     {
771 +         unsigned int md_size_u;
772 +         /* Chop the digest off the end :-> */
773 +         EVP_MD_CTX_init(&md_ctx);
774 +
775 +         EVP_MD_CTX_copy_ex( &md_ctx,hash);
776 +         EVP_DigestUpdate(&md_ctx,mac_sec,md_size);
777 +         EVP_DigestUpdate(&md_ctx,ssl3_pad_1,npad);
778 +         EVP_DigestUpdate(&md_ctx,seq,8);
779 +         rec_char=rec->type;
780 +         EVP_DigestUpdate(&md_ctx,&rec_char,1);
781 +         p=md;
782 +         s2n(rec->length,p);
783 +         EVP_DigestUpdate(&md_ctx,md,2);
784 +         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

```
384 384     SSL3_RECORD *rr;
385 385     unsigned int mac_size;
386 386     int decryption;
387 387     int r;
388 388     rrr = &(s->s3->
417 418     enc_err = s->
419 419     if (enc_err <
419 420     {
420 421     /* de
421 422     if (e
422 423     if (!send &&
423 424     EVP_CIPHER
424 425     ssl3_cbc;
425 426     }
426 426     goto
427 427     /* Pe
427 428     /* da
428 429     /* ch
429 430     /* ch
430 431     /* ch
431 432     /* ch
432 433     /* ch
433 434     /* ch
434 435     /* ch
435 436     /* ch
436 437     /* ch
437 438     /* ch
438 439     /* ch
439 440     /* ch
440 441     /* ch
441 442     /* ch
442 443     /* ch
443 444     /* ch
444 445     /* ch
445 446     /* ch
446 447     /* ch
447 448     /* ch
448 449     /* ch
449 450     /* ch
450 451     /* ch
451 452     /* ch
452 453     /* ch
453 454     /* ch
454 455     /* ch
455 456     /* ch
456 457     /* ch
457 458     /* ch
458 459     /* ch
459 460     /* ch
460 461     /* ch
461 462     /* ch
462 463     /* ch
463 464     /* ch
464 465     /* ch
465 466     /* ch
466 467     /* ch
467 468     /* ch
468 469     /* ch
469 470     /* ch
470 471     /* ch
471 472     /* ch
472 473     /* ch
473 474     /* ch
474 475     /* ch
475 476     /* ch
476 477     /* ch
477 478     /* ch
478 479     /* ch
479 480     /* ch
480 481     /* ch
481 482     /* ch
482 483     /* ch
483 484     /* ch
484 485     /* ch
485 486     /* ch
486 487     /* ch
487 488     /* ch
488 489     /* ch
489 490     /* ch
490 491     /* ch
491 492     /* ch
492 493     /* ch
493 494     /* ch
494 495     /* ch
495 496     /* ch
496 497     /* ch
497 498     /* ch
498 499     /* ch
499 500     /* ch
500 501     /* ch
501 502     /* ch
502 503     /* ch
503 504     /* ch
504 505     /* ch
505 506     /* ch
506 507     /* ch
507 508     /* ch
508 509     /* ch
509 510     /* ch
510 511     /* ch
511 512     /* ch
512 513     /* ch
513 514     /* ch
514 515     /* ch
515 516     /* ch
516 517     /* ch
517 518     /* ch
518 519     /* ch
519 520     /* ch
520 521     /* ch
521 522     /* ch
522 523     /* ch
523 524     /* ch
524 525     /* ch
525 526     /* ch
526 527     /* ch
527 528     /* ch
528 529     /* ch
529 530     /* ch
530 531     /* ch
531 532     /* ch
532 533     /* ch
533 534     /* ch
534 535     /* ch
535 536     /* ch
536 537     /* ch
537 538     /* ch
538 539     /* ch
539 540     /* ch
540 541     /* ch
541 542     /* ch
542 543     /* ch
543 544     /* ch
544 545     /* ch
545 546     /* ch
546 547     /* ch
547 548     /* ch
548 549     /* ch
549 550     /* ch
550 551     /* ch
551 552     /* ch
552 553     /* ch
553 554     /* ch
554 555     /* ch
555 556     /* ch
556 557     /* ch
557 558     /* ch
558 559     /* ch
559 560     /* ch
560 561     /* ch
561 562     /* ch
562 563     /* ch
563 564     /* ch
564 565     /* ch
565 566     /* ch
566 567     /* ch
567 568     /* ch
568 569     /* ch
569 570     /* ch
570 571     /* ch
571 572     /* ch
572 573     /* ch
573 574     /* ch
574 575     /* ch
575 576     /* ch
576 577     /* ch
577 578     /* ch
578 579     /* ch
579 580     /* ch
580 581     /* ch
581 582     /* ch
582 583     /* ch
583 584     /* ch
584 585     /* ch
585 586     /* ch
586 587     /* ch
587 588     /* ch
588 589     /* ch
589 590     /* ch
590 591     /* ch
591 592     /* ch
592 593     /* ch
593 594     /* ch
594 595     /* ch
595 596     /* ch
596 597     /* ch
597 598     /* ch
598 599     /* ch
599 600     /* ch
600 601     /* ch
601 602     /* ch
602 603     /* ch
603 604     /* ch
604 605     /* ch
605 606     /* ch
606 607     /* ch
607 608     /* ch
608 609     /* ch
609 610     /* ch
610 611     /* ch
611 612     /* ch
612 613     /* ch
613 614     /* ch
614 615     /* ch
615 616     /* ch
616 617     /* ch
617 618     /* ch
618 619     /* ch
619 620     /* ch
620 621     /* ch
621 622     /* ch
622 623     /* ch
623 624     /* ch
624 625     /* ch
625 626     /* ch
626 627     /* ch
627 628     /* ch
628 629     /* ch
629 630     /* ch
630 631     /* ch
631 632     /* ch
632 633     /* ch
633 634     /* ch
634 635     /* ch
635 636     /* ch
636 637     /* ch
637 638     /* ch
638 639     /* ch
639 640     /* ch
640 641     /* ch
641 642     /* ch
642 643     /* ch
643 644     /* ch
644 645     /* ch
645 646     /* ch
646 647     /* ch
647 648     /* ch
648 649     /* ch
649 650     /* ch
650 651     /* ch
651 652     /* ch
652 653     /* ch
653 654     /* ch
654 655     /* ch
655 656     /* ch
656 657     /* ch
657 658     /* ch
658 659     /* ch
659 660     /* ch
660 661     /* ch
661 662     /* ch
662 663     /* ch
663 664     /* ch
664 665     /* ch
665 666     /* ch
666 667     /* ch
667 668     /* ch
668 669     /* ch
669 670     /* ch
670 671     /* ch
671 672     /* ch
672 673     /* ch
673 674     /* ch
674 675     /* ch
675 676     /* ch
676 677     /* ch
677 678     /* ch
678 679     /* ch
679 680     /* ch
680 681     /* ch
681 682     /* ch
682 683     /* ch
683 684     /* ch
684 685     /* ch
685 686     /* ch
686 687     /* ch
687 688     /* ch
688 689     /* ch
689 690     /* ch
690 691     /* ch
691 692     /* ch
692 693     /* ch
693 694     /* ch
694 695     /* ch
695 696     /* ch
696 697     /* ch
697 698     /* ch
698 699     /* ch
699 700     /* ch
700 701     /* ch
701 702     /* ch
702 703     /* ch
703 704     /* ch
704 705     /* ch
705 706     /* ch
706 707     /* ch
707 708     /* ch
708 709     /* ch
709 710     /* ch
710 711     /* ch
711 712     /* ch
712 713     /* ch
713 714     /* ch
714 715     /* ch
715 716     /* ch
716 717     /* ch
717 718     /* ch
718 719     /* ch
719 720     /* ch
720 721     /* ch
721 722     /* ch
722 723     /* ch
723 724     /* ch
724 725     /* ch
725 726     /* ch
726 727     /* ch
727 728     /* ch
728 729     /* ch
729 730     /* ch
730 731     /* ch
731 732     /* ch
732 733     /* ch
733 734     /* ch
734 735     /* ch
735 736     /* ch
736 737     /* ch
737 738     /* ch
738 739     /* ch
739 740     /* ch
740 741     /* ch
741 742     /* ch
742 743     /* ch
743 744     /* ch
744 745     /* ch
745 746     /* ch
746 747     /* ch
747 748     /* ch
748 749     /* ch
749 750     /* ch
750 751     /* ch
751 752     /* ch
752 753     /* ch
753 754     /* ch
754 755     /* ch
755 756     /* ch
756 757     /* ch
757 758     /* ch
758 759     /* ch
759 760     /* ch
760 761     /* ch
761 762     /* ch
762 763     /* ch
763 764     /* ch
764 765     /* ch
765 766     /* ch
766 767     /* ch
767 768     /* ch
768 769     /* ch
769 770     /* ch
770 771     /* ch
771 772     /* ch
772 773     /* ch
773 774     /* ch
774 775     /* ch
775 776     /* ch
776 777     /* ch
777 778     /* ch
778 779     /* ch
779 780     /* ch
780 781     /* ch
781 782     /* ch
782 783     /* ch
783 784     /* ch
784 785     /* ch
785 786     /* ch
786 787     /* ch
787 788     /* ch
788 789     /* ch
789 790     /* ch
790 791     /* ch
791 792     /* ch
792 793     /* ch
793 794     /* ch
794 795     /* ch
795 796     /* ch
796 797     /* ch
797 798     /* ch
798 799     /* ch
799 800     /* ch
800 801     /* ch
801 802     /* ch
802 803     /* ch
803 804     /* ch
804 805     /* ch
805 806     /* ch
806 807     /* ch
807 808     /* ch
808 809     /* ch
809 810     /* ch
810 811     /* ch
811 812     /* ch
812 813     /* ch
813 814     /* ch
814 815     /* ch
815 816     /* ch
816 817     /* ch
817 818     /* ch
818 819     /* ch
819 820     /* ch
820 821     /* ch
821 822     /* ch
822 823     /* ch
823 824     /* ch
824 825     /* ch
825 826     /* ch
826 827     /* ch
827 828     /* ch
828 829     /* ch
829 830     /* ch
830 831     /* ch
831 832     /* ch
832 833     /* ch
833 834     /* ch
834 835     /* ch
835 836     /* ch
836 837     /* ch
837 838     /* ch
838 839     /* ch
839 840     /* ch
840 841     /* ch
841 842     /* ch
842 843     /* ch
843 844     /* ch
844 845     /* ch
845 846     /* ch
846 847     /* ch
847 848     /* ch
848 849     /* ch
849 850     /* ch
850 851     /* ch
851 852     /* ch
852 853     /* ch
853 854     /* ch
854 855     /* ch
855 856     /* ch
856 857     /* ch
857 858     /* ch
858 859     /* ch
859 860     /* ch
860 861     /* ch
861 862     /* ch
862 863     /* ch
863 864     /* ch
864 865     /* ch
865 866     /* ch
866 867     /* ch
867 868     /* ch
868 869     /* ch
869 870     /* ch
870 871     /* ch
871 872     /* ch
872 873     /* ch
873 874     /* ch
874 875     /* ch
875 876     /* ch
876 877     /* ch
877 878     /* ch
878 879     /* ch
879 880     /* ch
880 881     /* ch
881 882     /* ch
882 883     /* ch
883 884     /* ch
884 885     /* ch
885 886     /* ch
886 887     /* ch
887 888     /* ch
888 889     /* ch
889 890     /* ch
890 891     /* ch
891 892     /* ch
892 893     /* ch
893 894     /* ch
894 895     /* ch
895 896     /* ch
896 897     /* ch
897 898     /* ch
898 899     /* ch
899 900     /* ch
900 901     /* ch
901 902     /* ch
902 903     /* ch
903 904     /* ch
904 905     /* ch
905 906     /* ch
906 907     /* ch
907 908     /* ch
908 909     /* ch
909 910     /* ch
910 911     /* ch
911 912     /* ch
912 913     /* ch
913 914     /* ch
914 915     /* ch
915 916     /* ch
916 917     /* ch
917 918     /* ch
918 919     /* ch
919 920     /* ch
920 921     /* ch
921 922     /* ch
922 923     /* ch
923 924     /* ch
924 925     /* ch
925 926     /* ch
926 927     /* ch
927 928     /* ch
928 929     /* ch
929 930     /* ch
930 931     /* ch
931 932     /* ch
932 933     /* ch
933 934     /* ch
934 935     /* ch
935 936     /* ch
936 937     /* ch
937 938     /* ch
938 939     /* ch
939 940     /* ch
940 941     /* ch
941 942     /* ch
942 943     /* ch
943 944     /* ch
944 945     /* ch
945 946     /* ch
946 947     /* ch
947 948     /* ch
948 949     /* ch
949 950     /* ch
950 951     /* ch
951 952     /* ch
952 953     /* ch
953 954     /* ch
954 955     /* ch
955 956     /* ch
956 957     /* ch
957 958     /* ch
958 959     /* ch
959 960     /* ch
960 961     /* ch
961 962     /* ch
962 963     /* ch
963 964     /* ch
964 965     /* ch
965 966     /* ch
966 967     /* ch
967 968     /* ch
968 969     /* ch
969 970     /* ch
970 971     /* ch
971 972     /* ch
972 973     /* ch
973 974     /* ch
974 975     /* ch
975 976     /* ch
976 977     /* ch
977 978     /* ch
978 979     /* ch
979 980     /* ch
980 981     /* ch
981 982     /* ch
982 983     /* ch
983 984     /* ch
984 985     /* ch
985 986     /* ch
986 987     /* ch
987 988     /* ch
988 989     /* ch
989 990     /* ch
990 991     /* ch
991 992     /* ch
992 993     /* ch
993 994     /* ch
994 995     /* ch
995 996     /* ch
996 997     /* ch
997 998     /* ch
998 999     /* ch
999 1000     /* ch
```

Further refinements  
Decryption path has no more measurable timing differences

# Error-prone in practice

CVE-2016-2107

Somorovsky. “Curious padding oracle in OpenSSL.”

```
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;
```

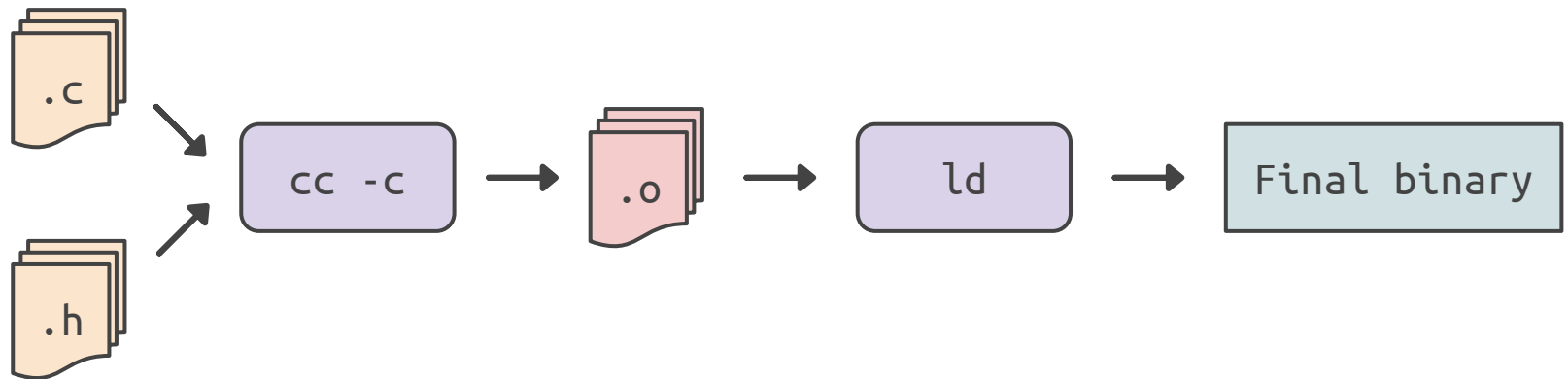
```
488 489 #endif
489 490
490 491 rr->i
491 492 ins->
492 493 if (l
493 494
494 495
495 496
496 497
497 498
498 499
499 500
500 501
501 502
502 503
503 504
504 505
505 506
506 507
507 508
508 509
509 510
510 511
511 512
512 513
513 514
514 515
515 516
516 517
517 518
518 519
519 520
520 521
521 522
522 523
523 524
524 525
525 526
526 527
527 528
528 529
529 530
530 531
531 532
532 533
533 534
534 535
535 536
536 537
537 538
538 539
539 540
540 541
541 542
542 543
543 544
544 545
545 546
546 547
547 548
548 549
549 550
550 551
551 552
552 553
553 554
554 555
555 556
556 557
557 558
558 559
559 560
560 561
561 562
562 563
563 564
564 565
565 566
566 567
567 568
568 569
569 570
570 571
571 572
572 573
573 574
574 575
575 576
576 577
577 578
578 579
579 580
580 581
581 582
582 583
583 584
584 585
585 586
586 587
587 588
588 589
589 590
590 591
591 592
592 593
593 594
594 595
595 596
596 597
597 598
598 599
599 600
600 601
601 602
602 603
603 604
604 605
605 606
606 607
607 608
608 609
609 610
610 611
611 612
612 613
613 614
614 615
615 616
616 617
617 618
618 619
619 620
620 621
621 622
622 623
623 624
624 625
625 626
626 627
627 628
628 629
629 630
630 631
631 632
632 633
633 634
634 635
635 636
636 637
637 638
638 639
639 640
640 641
641 642
642 643
643 644
644 645
645 646
646 647
647 648
648 649
649 650
650 651
651 652
652 653
653 654
654 655
655 656
656 657
657 658
658 659
659 660
660 661
661 662
662 663
663 664
664 665
665 666
666 667
667 668
668 669
669 670
670 671
671 672
672 673
673 674
674 675
675 676
676 677
677 678
678 679
679 680
680 681
681 682
682 683
683 684
684 685
685 686
686 687
687 688
688 689
689 690
690 691
691 692
692 693
693 694
694 695
695 696
696 697
697 698
698 699
699 700
700 701
701 702
702 703
703 704
704 705
705 706
706 707
707 708
708 709
709 710
710 711
711 712
712 713
713 714
714 715
715 716
716 717
717 718
718 719
719 720
720 721
721 722
722 723
723 724
724 725
725 726
726 727
727 728
728 729
729 730
730 731
731 732
732 733
733 734
734 735
735 736
736 737
737 738
738 739
739 740
740 741
741 742
742 743
743 744
744 745
745 746
746 747
747 748
748 749
749 750
750 751
751 752
752 753
753 754
754 755
755 756
756 757
757 758
758 759
759 760
760 761
761 762
762 763
763 764
764 765
765 766
766 767
767 768
768 769
769 770
770 771
771 772
772 773
773 774
774 775
775 776
776 777
777 778
778 779
779 780
780 781
781 782
782 783
783 784
784 785
785 786
786 787
787 788
788 789
789 790
790 791
791 792
792 793
793 794
794 795
795 796
796 797
797 798
798 799
799 800
800 801
801 802
802 803
803 804
804 805
805 806
806 807
807 808
808 809
809 810
810 811
811 812
812 813
813 814
814 815
815 816
816 817
817 818
818 819
819 820
820 821
821 822
822 823
823 824
824 825
825 826
826 827
827 828
828 829
829 830
830 831
831 832
832 833
833 834
834 835
835 836
836 837
837 838
838 839
839 840
840 841
841 842
842 843
843 844
844 845
845 846
846 847
847 848
848 849
849 850
850 851
851 852
852 853
853 854
854 855
855 856
856 857
857 858
858 859
859 860
860 861
861 862
862 863
863 864
864 865
865 866
866 867
867 868
868 869
869 870
870 871
871 872
872 873
873 874
874 875
875 876
876 877
877 878
878 879
879 880
880 881
881 882
882 883
883 884
884 885
885 886
886 887
887 888
888 889
889 890
890 891
891 892
892 893
893 894
894 895
895 896
896 897
897 898
898 899
899 900
900 901
901 902
902 903
903 904
904 905
905 906
906 907
907 908
908 909
909 910
910 911
911 912
912 913
913 914
914 915
915 916
916 917
917 918
918 919
919 920
920 921
921 922
922 923
923 924
924 925
925 926
926 927
927 928
928 929
929 930
930 931
931 932
932 933
933 934
934 935
935 936
936 937
937 938
938 939
939 940
940 941
941 942
942 943
943 944
944 945
945 946
946 947
947 948
948 949
949 950
950 951
951 952
952 953
953 954
954 955
955 956
956 957
957 958
958 959
959 960
960 961
961 962
962 963
963 964
964 965
965 966
966 967
967 968
968 969
969 970
970 971
971 972
972 973
973 974
974 975
975 976
976 977
977 978
978 979
979 980
980 981
981 982
982 983
983 984
984 985
985 986
986 987
987 988
988 989
989 990
990 991
991 992
992 993
993 994
994 995
995 996
996 997
997 998
998 999
999 1000
```

# This is what DSLs are for!

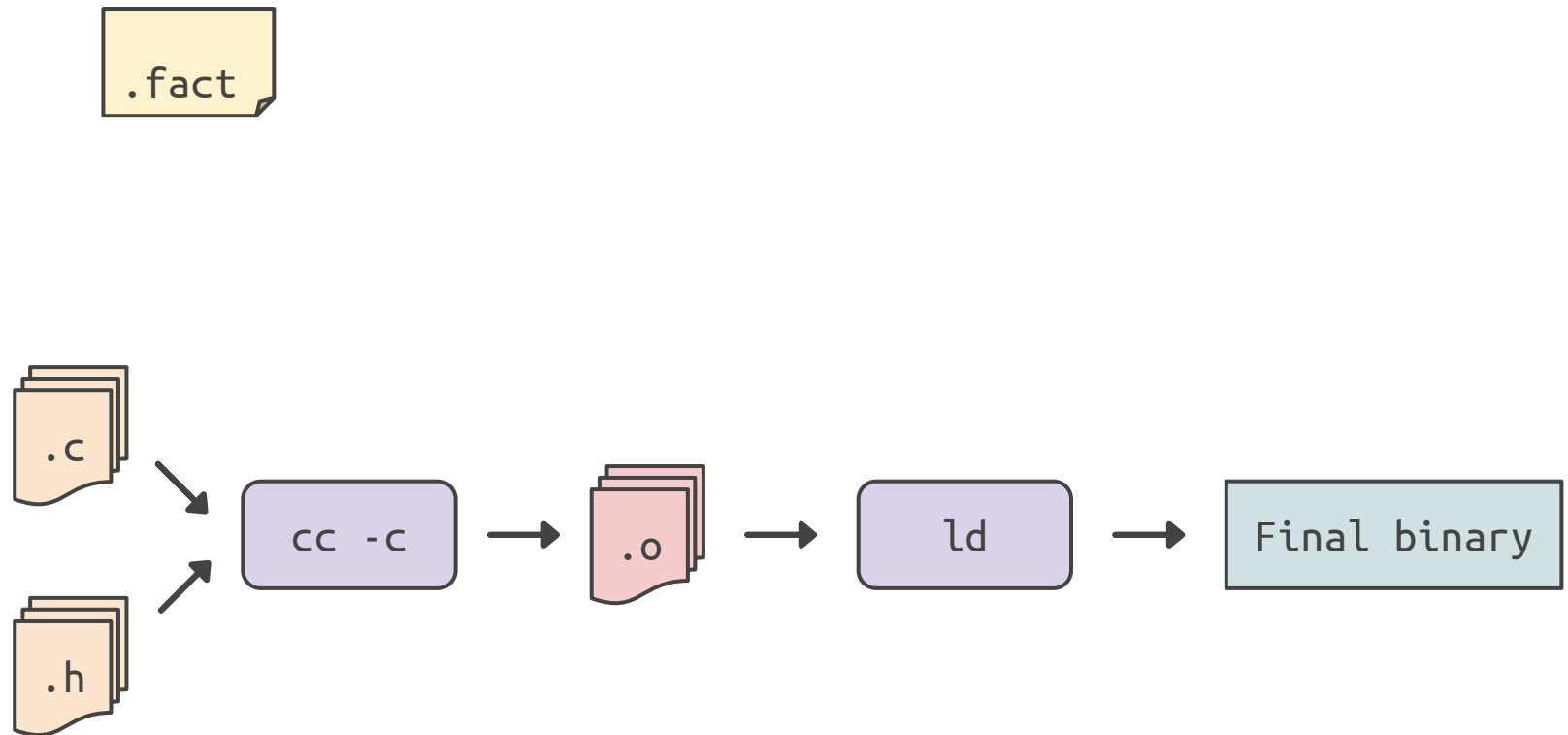
- Explicitly distinguish secret vs. public values
- Type system to prevent writing leaky code
- Compiler to transform high-level constructs to CT



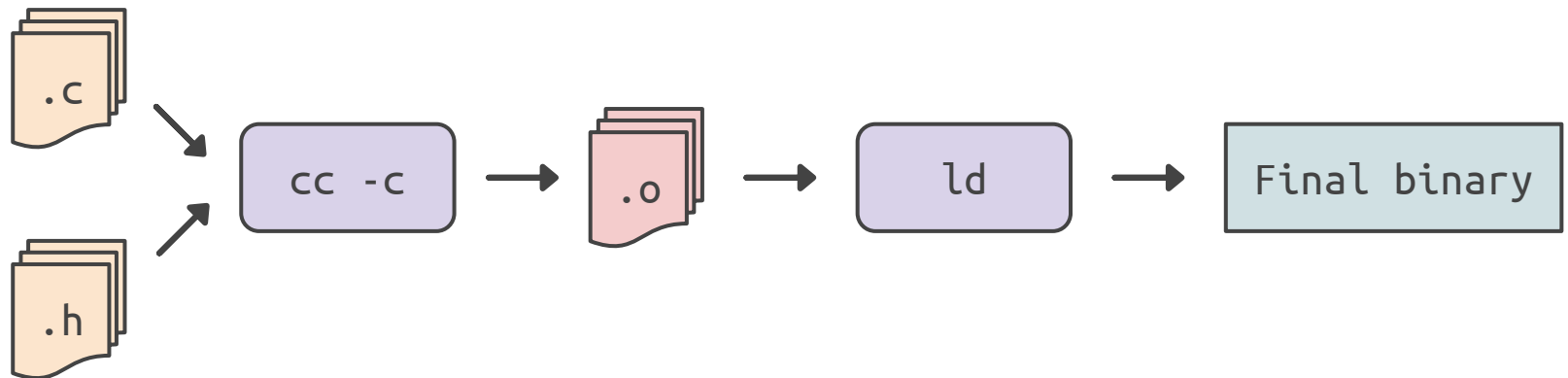
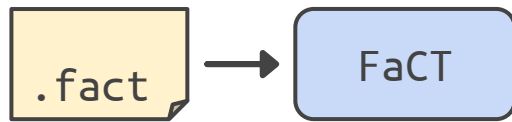
# FaCT



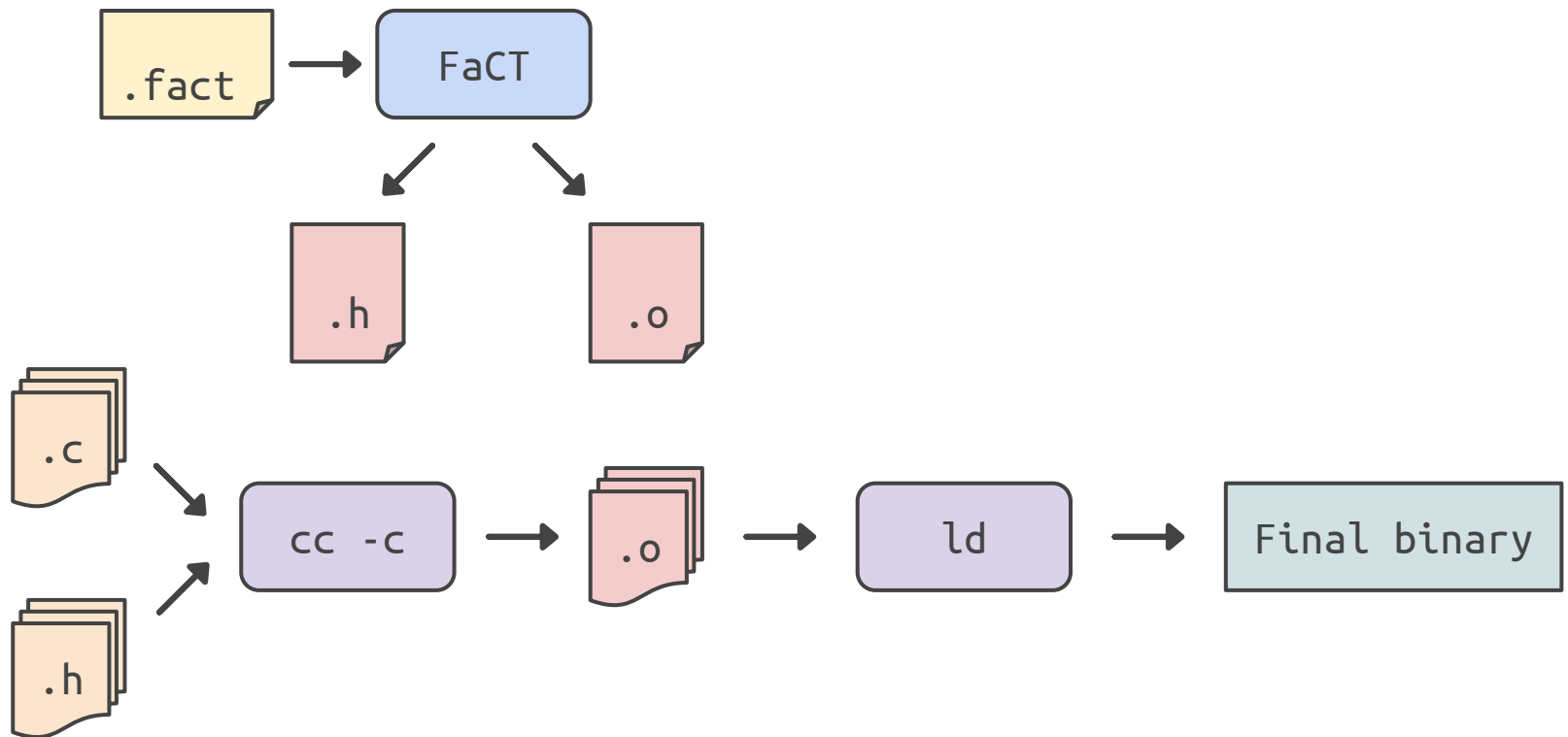
# FaCT



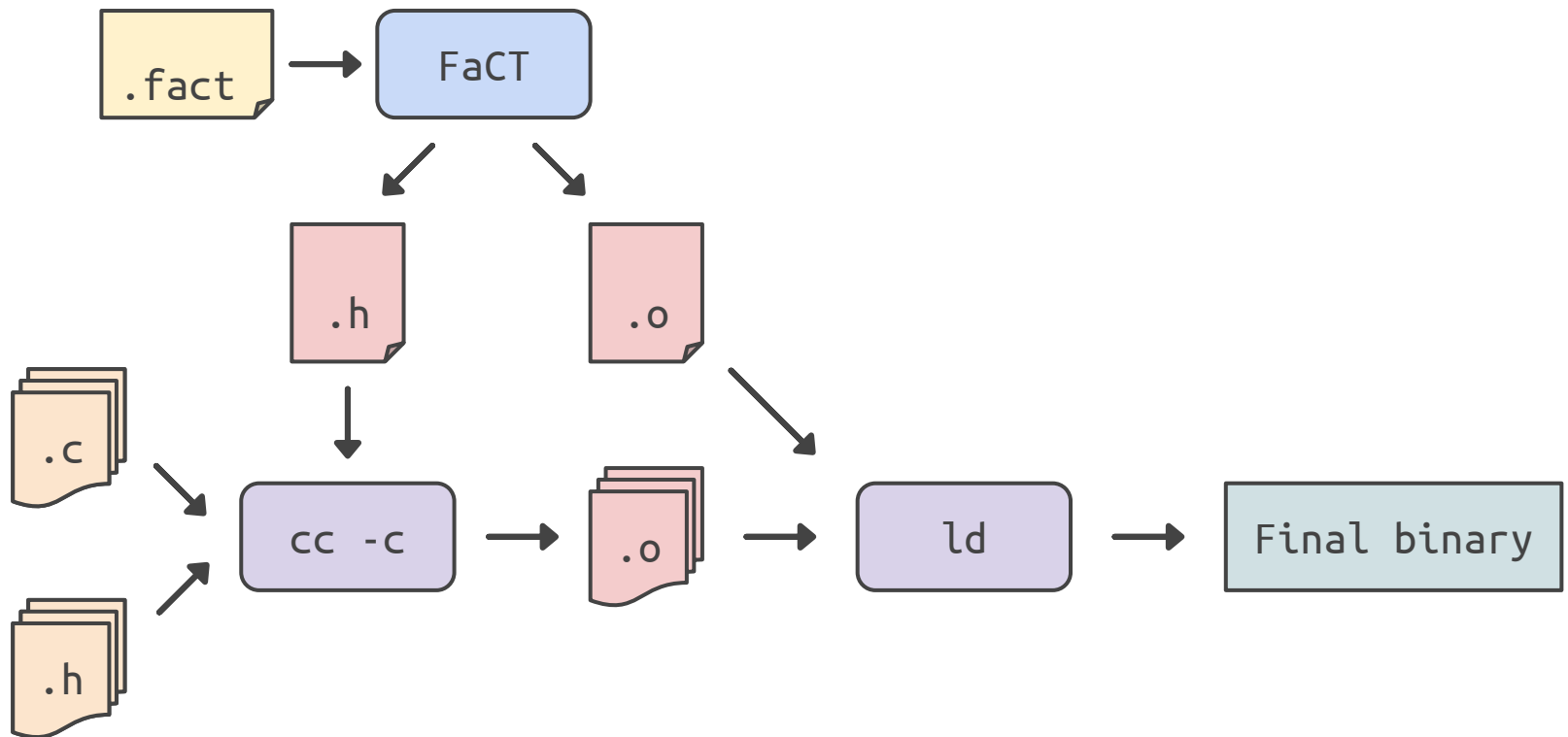
# FaCT



# FaCT



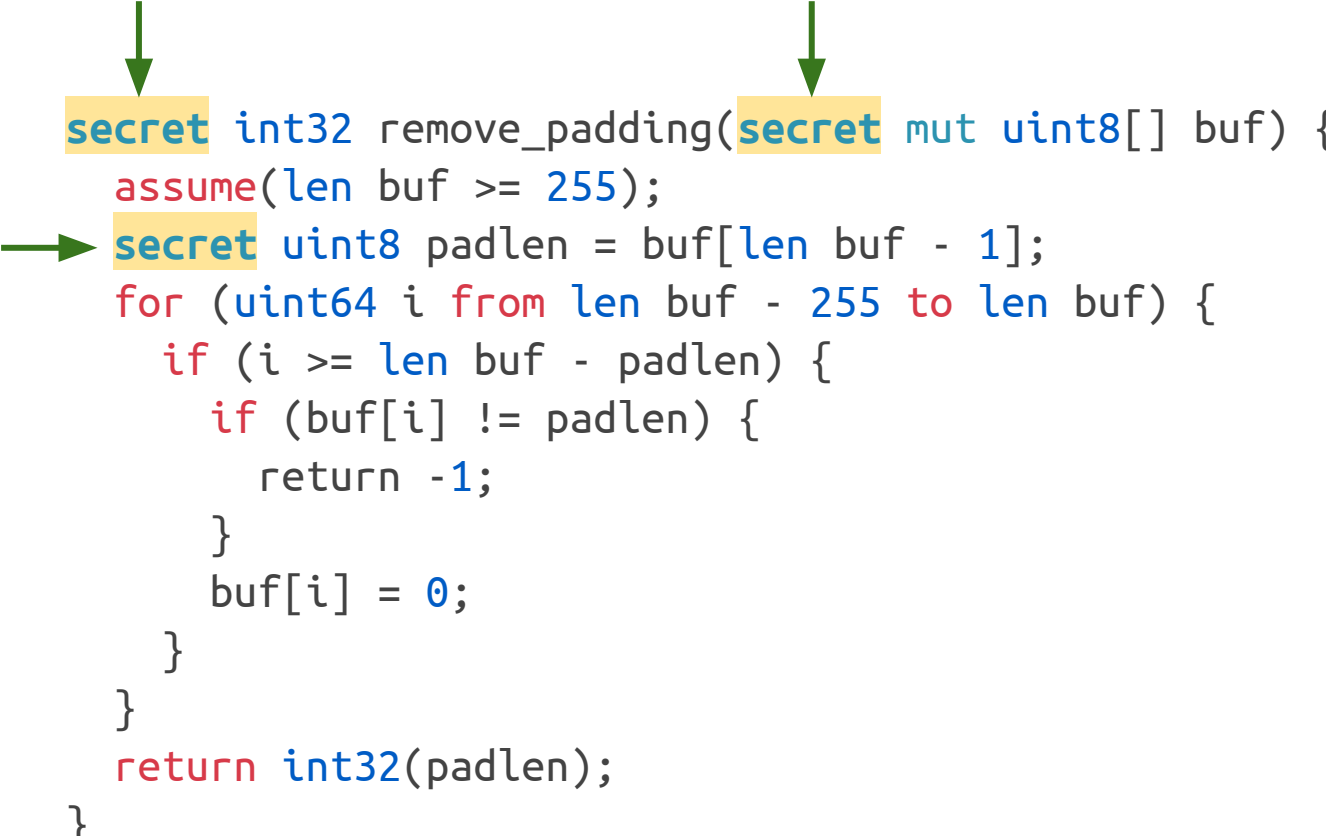
# FaCT



# 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);
}
```

# What does FaCT look like?

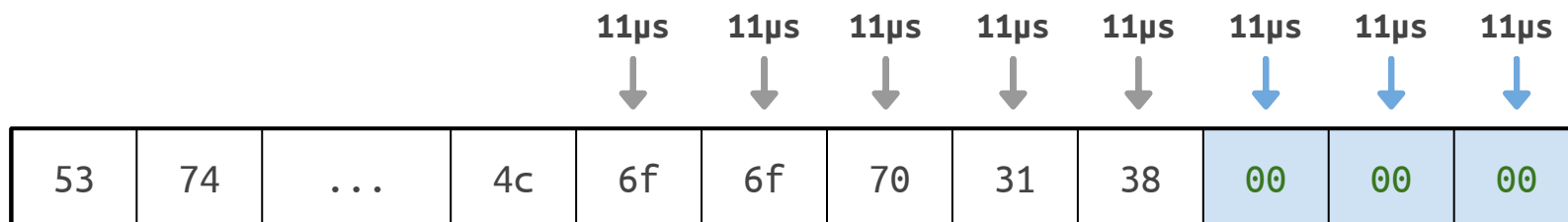
```
secret int32 remove_padding(secret mut uint8[] buf) {
  assume(len buf >= 255);
  secret uint8 padlen = buf[len buf - 1];
  for (uint64 i from len buf - 255 to len buf) {
    if (i >= len buf - padlen) {
      if (buf[i] != padlen) {
        return -1;
      }
      buf[i] = 0;
    }
  }
  return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

```
uint8_t b = buf[i];
uint32_t improper_index =
    -(i - (buflen - padlen) >> 31);
uint32_t matches_pad =
    -((b ^ padlen) - 1 >> 31);
ok &= matches_pad | improper_index;
b = improper_index & b;
buf[i] = b;
```

# What does FaCT look like?

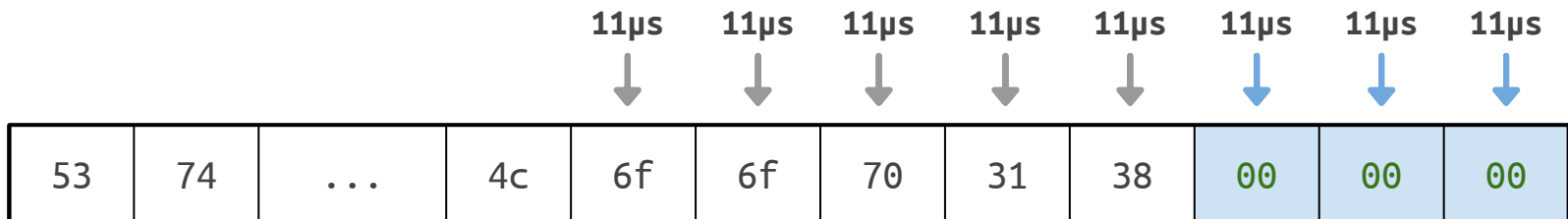
```
secret int32 remove_padding(secret mut uint8[] buf) {
  assume(len buf >= 255);
  secret uint8 padlen = buf[len buf - 1];
  for (uint64 i from len buf - 255 to len buf) {
    if (i >= len buf - padlen) {
      if (buf[i] != padlen) {
        return -1;
      }
      buf[i] = 0;
    }
  }
  return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

```
uint8_t b = buf[i];
uint32_t improper_index =
    -(i - (buflen - padlen) >> 31);
uint32_t matches_pad =
    -((b ^ padlen) - 1 >> 31);
ok &= matches_pad | improper_index;
b = improper_index & b;
buf[i] = b;
```



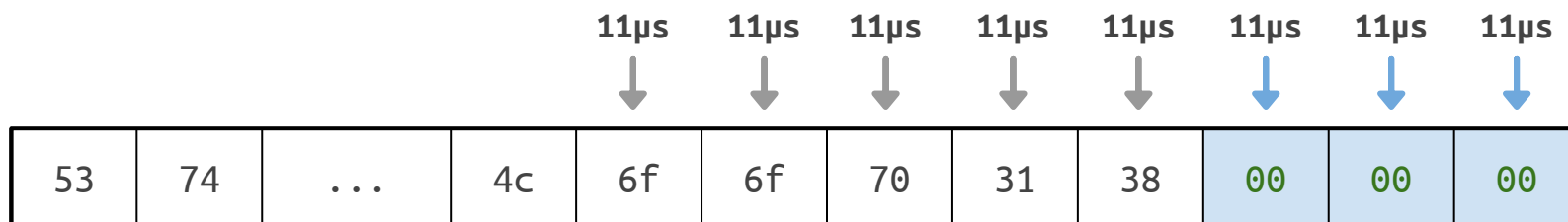
# What does FaCT look like?

```
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);  
}
```




# Labels ensure no leakage


- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions that leak:



# Labels ensure no leakage

- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions that leak:
  - Loop bounds

```
for (uint32 i from 0 to ) {  
    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 that leak:
  - Loop bounds

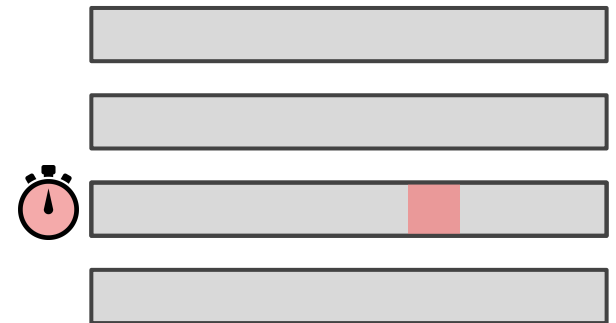
```
for (uint32 i from 0 to public_value) {  
    if (i < secret_value) {  
        do_operation();  
    }  
}
```

# Labels ensure no leakage

- No assignment from **secret** to **public**
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions that leak:
  - Loop bounds
  - Array indices

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

Cache lines

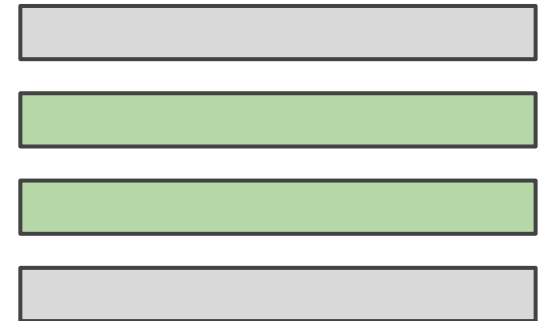


# Labels ensure no leakage

- No assignment from **secret** to **public**
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions that leak:
  - Loop bounds
  - Array indices

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

Cache lines



# Labels ensure no leakage

- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions that leak:
  - 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 that leak:
  - 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 that leak:
  - Loop bounds
  - Array indices
  - Variable-time instructions
  - Recursive calls

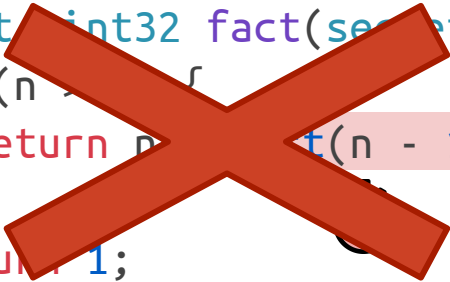
```
secret uint32 fact(secret uint32 n) {  
    if (n > 1) {  
        return n * fact(n - 1);  
    }  
    return 1;  
}
```



# Labels ensure no leakage

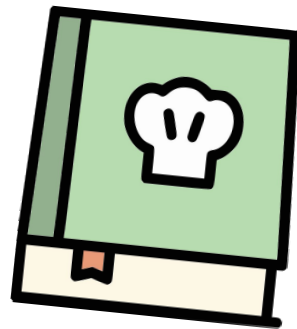
- No assignment from `secret` to `public`
- Type system tracks control flow label
  - Only transform secret control flow
- Prevent secret expressions that leak:
  - Loop bounds
  - Array indices
  - Variable-time instructions
  - Recursive calls

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

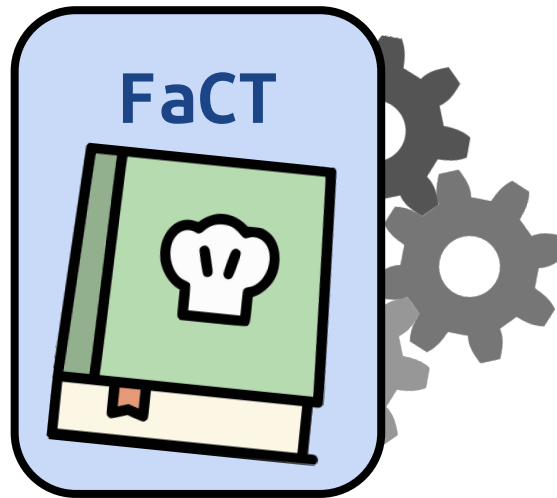




# Automatically transform code



# Automatically transform code



# Automatically transform code

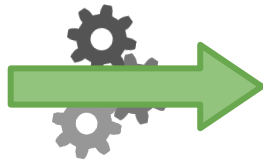
- Transform secret branches
- Keep track of static control flow

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

# Automatically transform code

- Transform secret branches
- 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

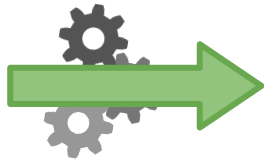
- Transform away early returns
- Keep track of current return state

```
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

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

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

# Automatically transform code

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

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



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

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




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



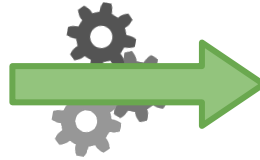

# Transformations are tricky

```
if (i < secret_index) {  
    buf[i] = 0;  
}
```



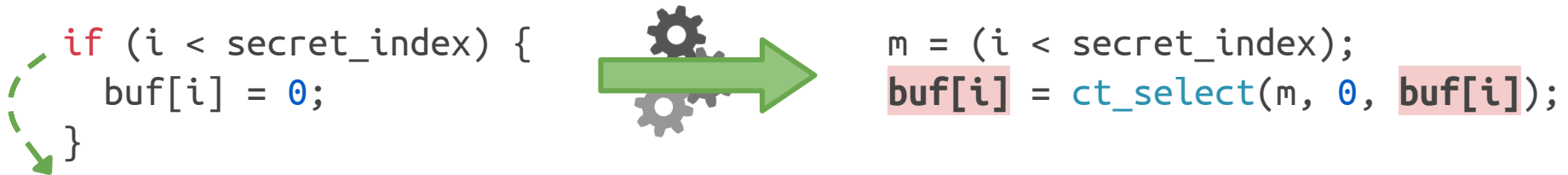
# Transformations are tricky

```
if (i < secret_index) {  
    buf[i] = 0;  
}
```



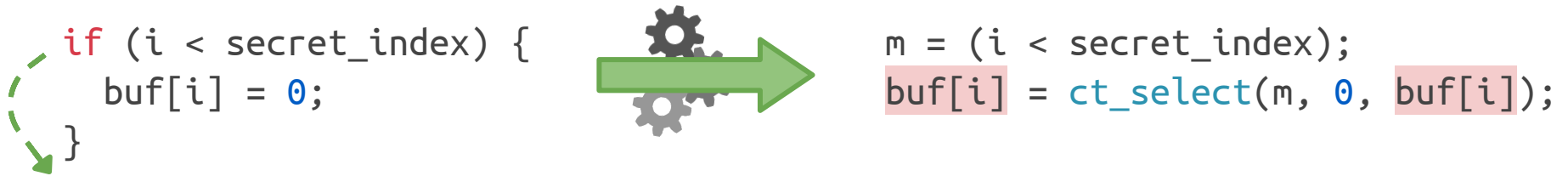
```
m = (i < secret_index);  
buf[i] = ct_select(m, 0, buf[i]);
```

# Transformations are tricky

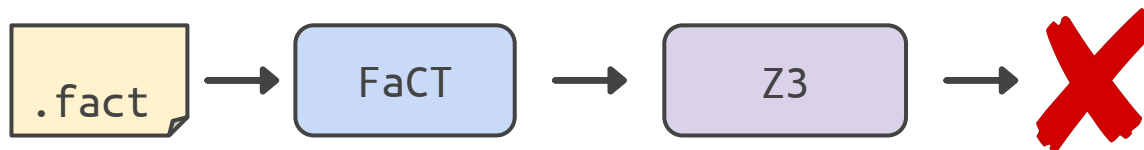


- **Problem:** secret if-statements always perform branches
  - Does not guard execution
  - Similar problem for secret early return

# Transformations are tricky

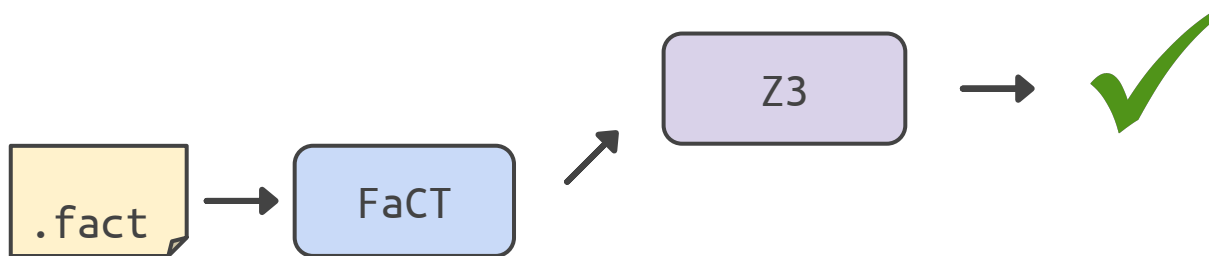


- **Problem:** secret if-statements always perform branches
  - Does not guard execution
  - Similar problem for secret early return
- **Solution:** disallow these programs!



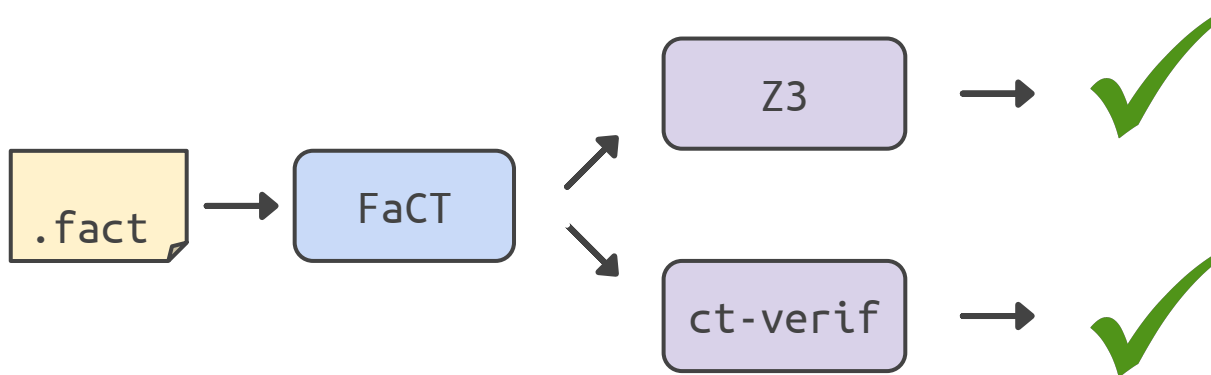
# Verifying constant-time code

- 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



# Interoperate with external code

- Generate function prototypes

```
/*public*/ uint8_t crypto_secretbox(  
    /*secret*/ uint8_t c[],  
    /*public*/ uint64_t c_len,  
    const /*public*/ uint8_t n[24],  
    const /*secret*/ uint8_t k[32]);
```

# Interoperate with external code

- Generate function prototypes
- Call external functions from FaCT

```
extern void  
aesni_cbc_encrypt(  
    secret mut uint8[] buf,  
    public uint64 buf_len,  
    AES_KEY key,  
    public int32 enc);
```



# Interoperate with external code

- Generate function prototypes
- Call external functions from FaCT
- Pass complex data structures

```
struct EVP_AES_HMAC_SHA1 {  
    AES_KEY ks;  
    SHA_CTX md;  
    public uint64 payload_length;  
    secret uint8[16] tls_aad;  
}
```

# Interoperate with external code

- Generate function prototypes
- Call external functions from FaCT
- Pass complex data structures
- Embed in other languages

```
import Language.FaCT.Inline

[fact|
  secret uint32 choose(
    secret bool b,
    secret uint32 x,
    secret uint32 y) {
    return b ? x : y;
  }
|]
```

# FaCT in practice

- Must be fast
- Must be usable

# FaCT in practice

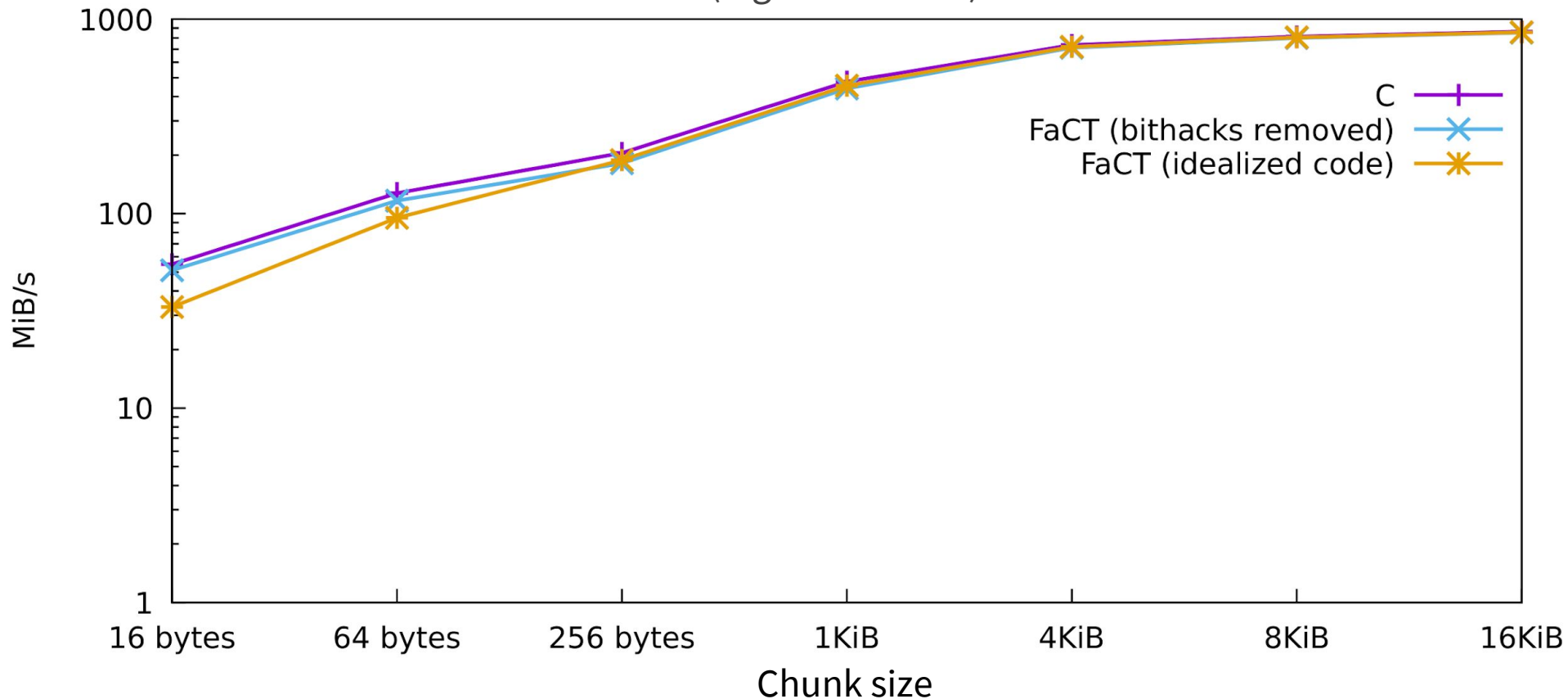
- Must be fast
- Must be usable
  
- Case studies:
  - libsodium
  - OpenSSL
  - mbedTLS
  - donna curve-25519

# FaCT in practice

- Must be fast
- Must be usable
  
- Case studies:
  - libsodium
  - OpenSSL
  - mbedTLS
  - donna curve-25519
- User study

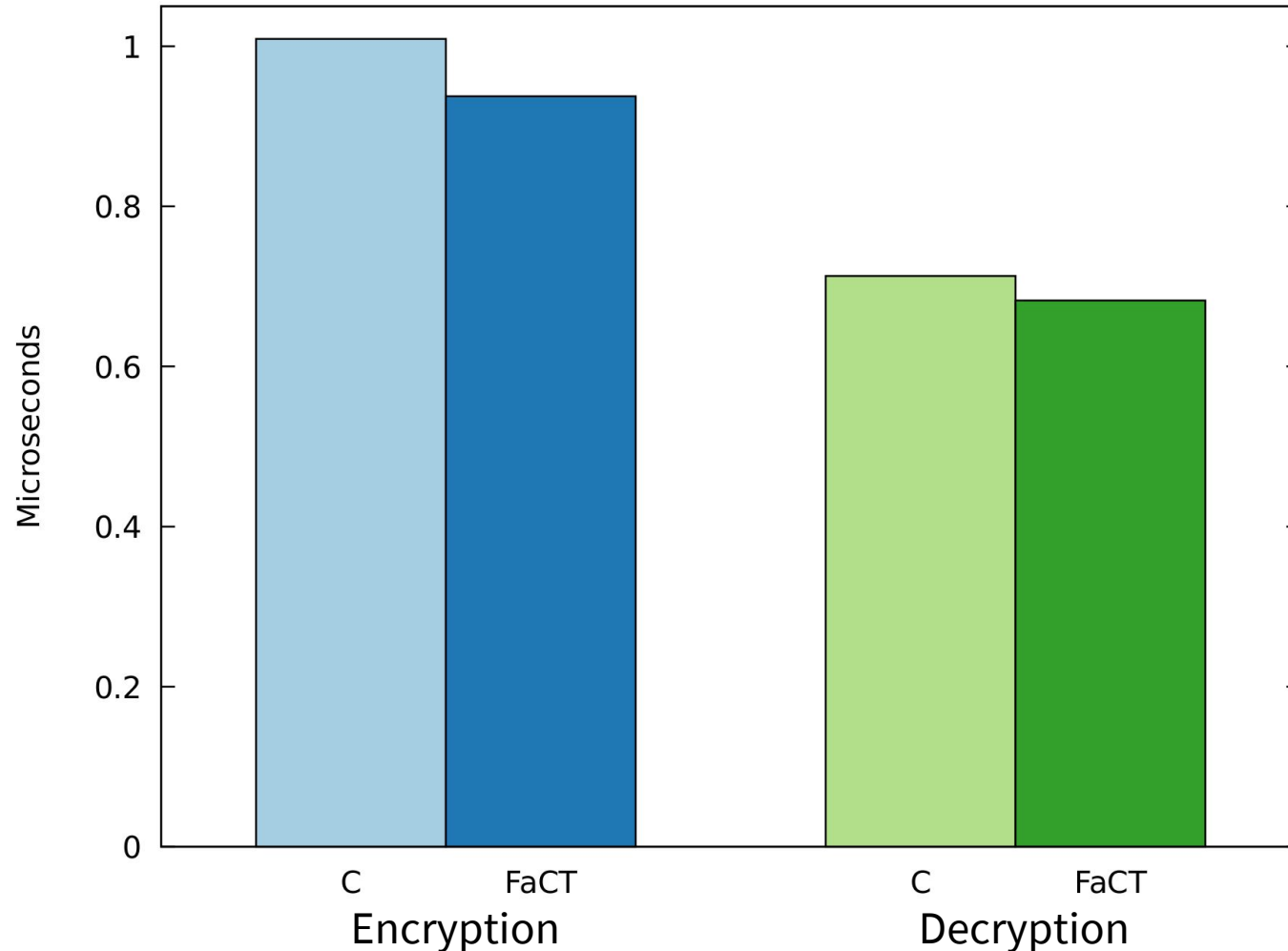
# Performance numbers

OpenSSL decryption throughput  
(higher is better)



# Performance numbers

libsodium secretbox benchmarks  
(lower is better)



# User study results

- Conducted user study on undergrad PL class
  - Understanding constant-time code (C vs. FaCT)
  - Writing constant-time code (C + ct-verif vs. FaCT)
- Results:
  - Students understood FaCT code better than C
  - More students successfully wrote FaCT code
  - Fewer security errors when writing FaCT
  - FaCT syntax tripped people up



# Future directions

- Add useful language features
- 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

