

# FaCT: A DSL for Timing-Sensitive Computation

**Sunjay Cauligi**, UC San Diego

Gary Soeller, Brian Johannesmeyer, Fraser Brown, Riad Wahby, John Renner,  
Benjamin Gregoire, Gilles Barthe, Ranjit Jhala, Deian Stefan

# What does this code do?

```
for (i = 0; i < n; i++) {  
    d |= x[i] ^ y[i];  
}  
return (1 & ((d - 1) >> 8)) - 1;
```

# What does *this* code do?

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```

# It compares two buffers.

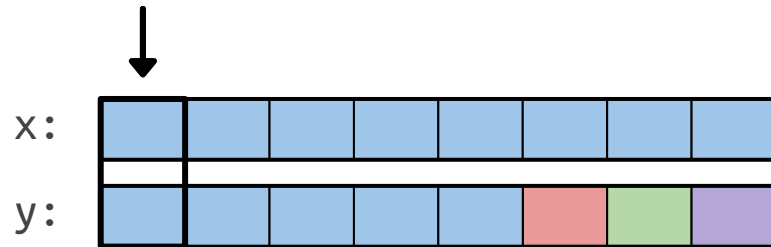
```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```

# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```

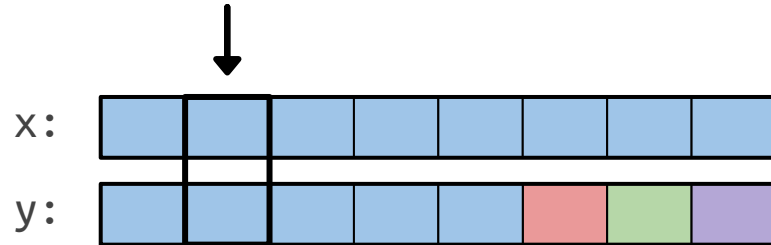
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```



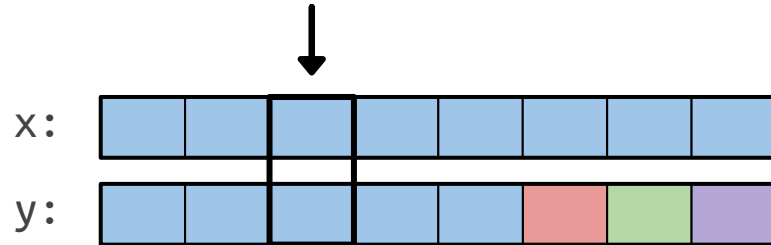
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```



# It compares two buffers.

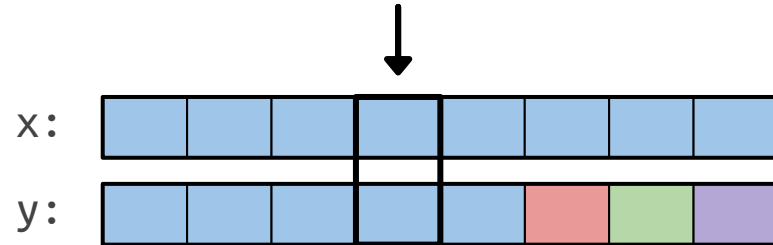
```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```





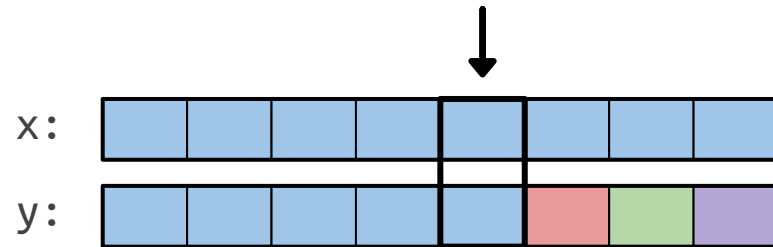
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```



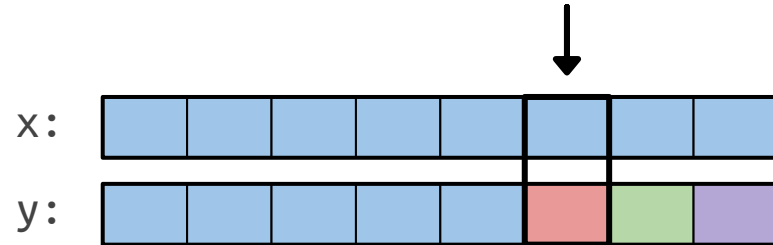
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```



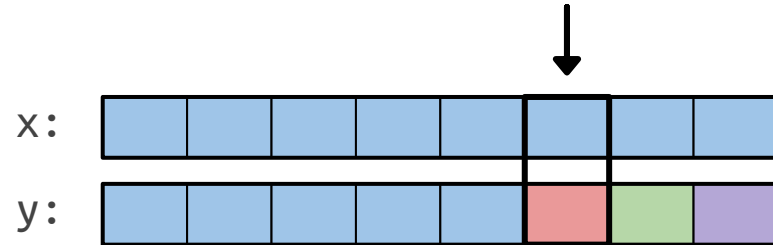
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```



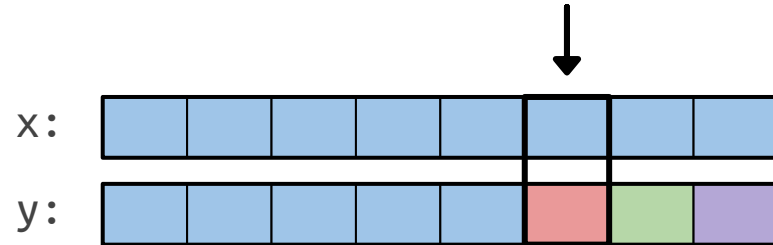
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i]) ←  
        return -1;  
}  
return 0;
```



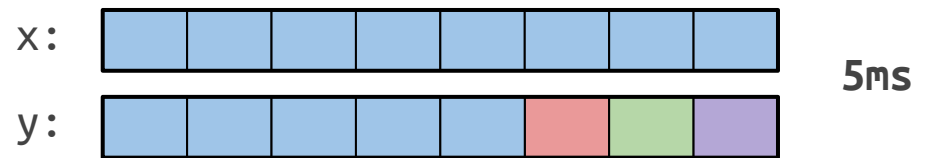
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1; ←  
}  
return 0;
```



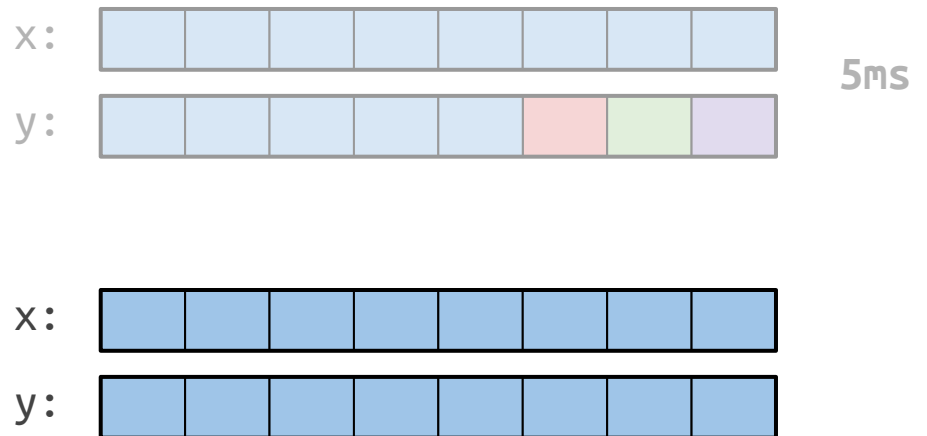
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```



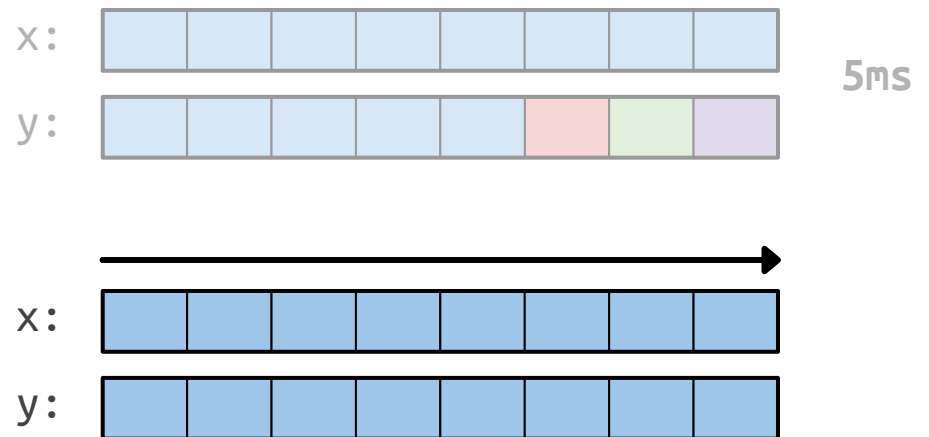
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```



# It compares two buffers.

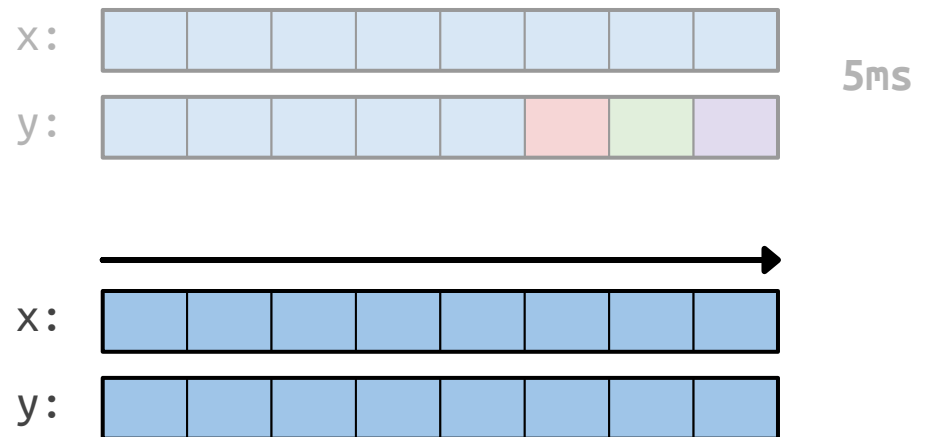
```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```





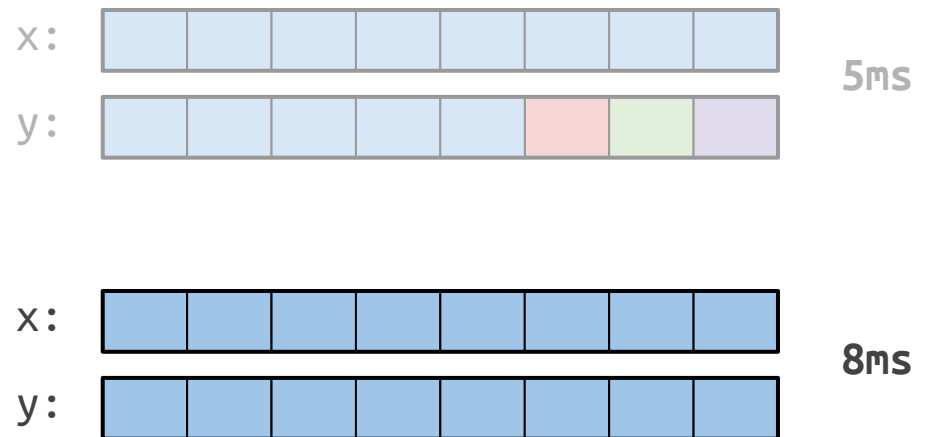
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0; ←
```



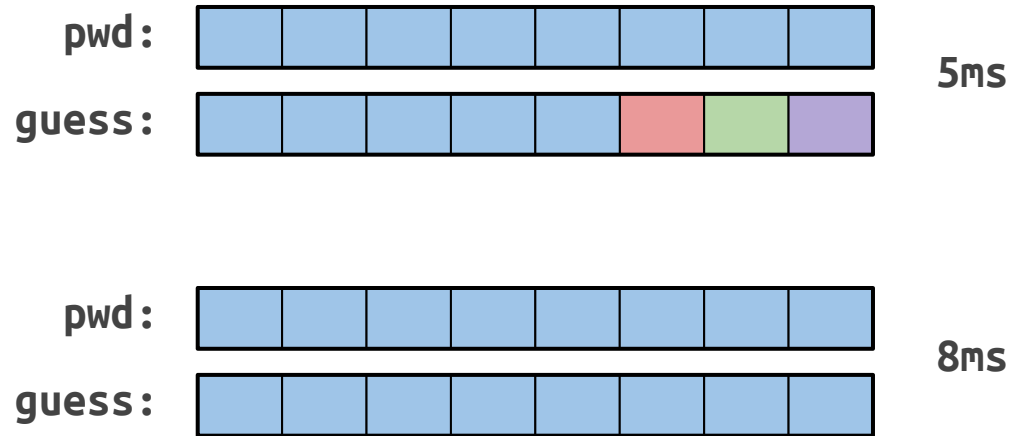
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```



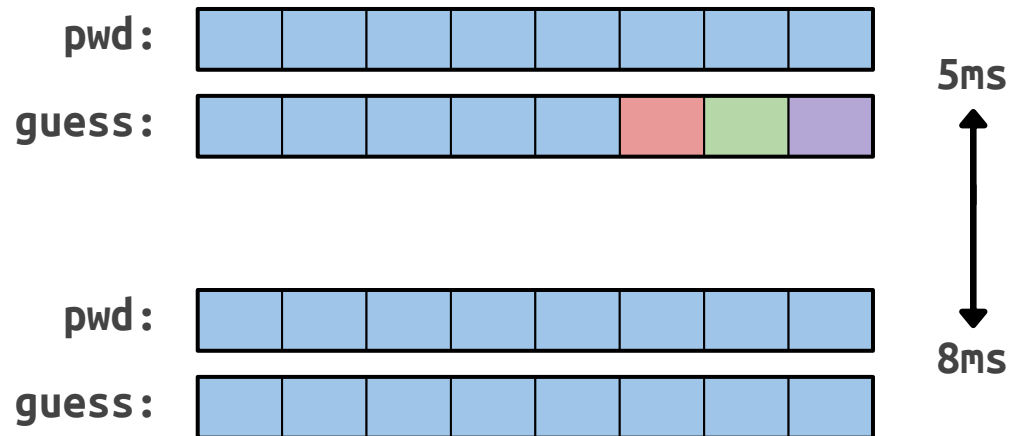
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```



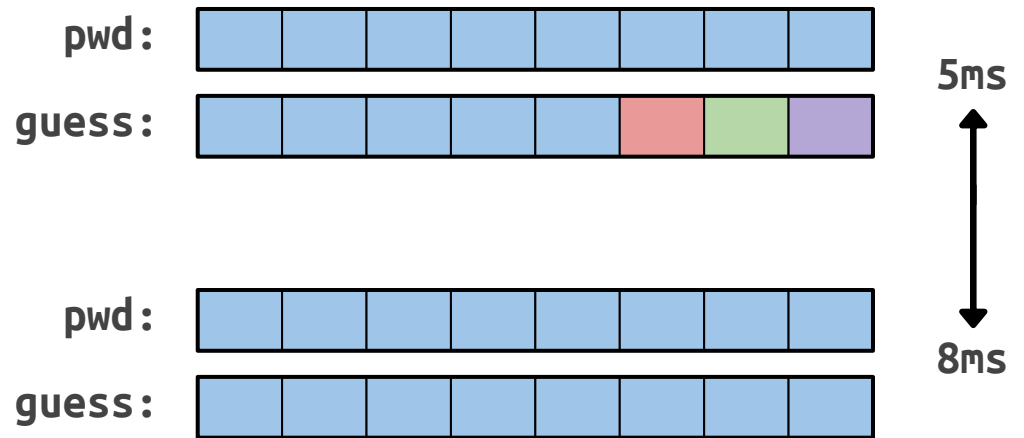
# It compares two buffers.

```
for (i = 0; i < n; i++) {  
  if (x[i] != y[i])  
    return -1;  
}  
return 0;
```



# It compares two buffers.

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```



**Exiting early** based on contents → leak!

# Must not exit early

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```

```
for (i = 0; i < n; i++) {  
    d |= x[i] ^ y[i];  
}  
return (1 & ((d - 1) >> 8)) - 1;
```

# Must not exit early

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```

```
for (i = 0; i < n; i++) {  
    d |= x[i] ^ y[i];  
}  
return (1 & ((d - 1) >> 8)) - 1;
```

# Must not exit early

## Constant-time code

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```

```
for (i = 0; i < n; i++) {  
    d |= x[i] ^ y[i];  
}  
return (1 & ((d - 1) >> 8)) - 1;
```



# Must not exit early

## Constant-time code

Timing is **independent of secrets**

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```

```
for (i = 0; i < n; i++) {  
    d |= x[i] ^ y[i];  
}  
return (1 & ((d - 1) >> 8)) - 1;
```

# Constant-time code is messy

```
for (j = 0; j < md_block_size; j++) {
    uint8_t b = data[j];
    uint8_t is_past_c = is_block_a & constant_time_ge_8_s(j, c);
    uint8_t is_past_cp1 = is_block_a & constant_time_ge_8_s(j, c + 1);
    b = constant_time_select_8(is_past_c, 0x80, b);
    b = b & ~is_past_cp1;
    b &= ~is_block_b | is_block_a;
    if (j >= md_block_size - md_length_size) {
        b = constant_time_select_8(is_block_b,
                                   length_bytes[j - (md_block_size - md_length_size)], b);
    }
    block[j] = b;
}
```

# Constant-time code is messy

**NOT READABLE!**

```
for (j = 0; j < md_block_size; j++)
    uint8_t b = data[j];
    uint8_t is_past_c = is_block_a & constant_time_ge_8_s(j, c),
    uint8_t is_past_cp1 = is_block_a & constant_time_ge_8_s(j, c + 1);
    b = constant_time_select_8(is_past_c, 0x80, b);
    b = b & ~is_past_cp1;
    b &= ~is_block_b | is_block_a;
    if (j >= md_block_size - md_length_size) {
        b = constant_time_select_8(is_block_b,
            length_bytes[j - (md_block_size - md_length_size)], b);
    }
    block[j] = b;
}
```

# Constant-time code is hard to write

## OpenSSL padding oracle attack

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

# Constant-time code is hard to write

```
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     rr = &(s->s3->rrec);
391 389
392 390     /* -417,13 +418,10 @@ dtls1_process_record(SSL *s)
393 390     enc_err = s->method->ssli3_enc->enc(s,0);
394 390     if (enc_err <= 0)
395 390     {
396 390     /* decryption failed, silently discard message */
397 390     if (enc_err < 0)
398 390     {
399 390     {
400 390     rr->length = 0;
401 390     s->packet_length = 0;
402 390     }
403 390     }
404 390     goto err;
405 390     /* To minimize information leaked via timing, we will always
406 390     * perform all computations before discarding the message.
407 390     */
408 390     decryption_failed_or_bad_record_mac = 1;
409 390     }
410 390
411 390     #ifdef TLS_DEBUG
412 390     /* -453,7 +451,7 @@ printf("\n");
413 390     SSLerr(SSL_F_DTLS1_PROCESS_RECORD,SSL_R_PRE_MAC_LENGTH_TOO_LONG);
414 390     goto f_err;
415 390
416 390     #else
417 390     goto err;
418 390     #endif
419 390     decryption_failed_or_bad_record_mac = 1;
420 390     #endif
421 390     /* check the MAC for rr->input (it's in mac_size bytes at the tail) */
422 390     /* -464,17 +462,25 @@ printf("\n");
423 390     SSLerr(SSL_F_DTLS1_PROCESS_RECORD,SSL_R_LENGTH_TOO_SHORT);
424 390     goto f_err;
425 390
426 390     #else
427 390     goto err;
428 390     #endif
429 390     decryption_failed_or_bad_record_mac = 1;
430 390     #endif
431 390     rr->length=mac_size;
432 390     !s->method->ssli3_enc->mac(s,md,0);
433 390     if (1 < 0 || memcmp(md,&(rr->data[rr->length]),mac_size) != 0)
434 390     {
435 390     goto err;
436 390     decryption_failed_or_bad_record_mac = 1;
437 390     }
438 390     }
439 390     }
440 390     if (decryption_failed_or_bad_record_mac)
441 390     {
442 390     /* decryption failed, silently discard message */
443 390     rr->length = 0;
444 390     s->packet_length = 0;
445 390     goto err;
446 390     }
447 390
448 390     /* f->length is now just compressed */
449 390     if (s->expand != NULL)
450 390     {
```

## OpenSSL padding oracle attack

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

# Constant-time code is hard to write

```
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         00 -417,13 +418,10 00)
390 389     enc_err = s->
391 389     if (enc_err <
392 389         {
393 389         }
394 389         /* de
395 389         if (e
396 389         }
397 389         }
398 389         }
399 389         }
400 389         }
401 389         }
402 389         }
403 389         }
404 389         }
405 389         }
406 389         }
407 389         }
408 389         }
409 389         }
410 389         }
411 389         }
412 389         }
413 389         }
414 389         }
415 389         }
416 389         }
417 389         }
418 389         }
419 389         }
420 389         }
421 389         }
422 389         }
423 389         }
424 389         }
425 389         }
426 389         }
427 389         }
428 389         }
429 389         }
430 389         }
431 389         }
432 389         }
433 389         }
434 389         }
435 389         }
436 389         }
437 389         }
438 389         }
439 389         }
440 389         }
441 389         }
442 389         }
443 389         }
444 389         }
445 389         }
446 389         }
447 389         }
448 389         }
449 389         }
450 389         }
451 389         }
452 389         }
453 389         }
454 389         }
455 389         }
456 389         }
457 389         }
458 389         }
459 389         }
460 389         }
461 389         }
462 389         }
463 389         }
464 389         }
465 389         }
466 389         }
467 389         }
468 389         }
469 389         }
470 389         }
471 389         }
472 389         }
473 389         }
474 389         }
475 389         }
476 389         }
477 389         }
478 389         }
479 389         }
480 389         }
481 389         }
482 389         }
483 389         }
484 389         }
485 389         }
486 389         }
487 389         }
488 389         }
489 389         }
490 389         }
491 389         }
492 389         }
493 389         }
494 389         }
495 389         }
496 389         }
497 389         }
498 389         }
499 389         }
500 389         }
501 389         }
502 389         }
503 389         }
504 389         }
505 389         }
506 389         }
507 389         }
508 389         }
509 389         }
510 389         }
511 389         }
512 389         }
513 389         }
514 389         }
515 389         }
516 389         }
517 389         }
518 389         }
519 389         }
520 389         }
521 389         }
522 389         }
523 389         }
524 389         }
525 389         }
526 389         }
527 389         }
528 389         }
529 389         }
530 389         }
531 389         }
532 389         }
533 389         }
534 389         }
535 389         }
536 389         }
537 389         }
538 389         }
539 389         }
540 389         }
541 389         }
542 389         }
543 389         }
544 389         }
545 389         }
546 389         }
547 389         }
548 389         }
549 389         }
550 389         }
551 389         }
552 389         }
553 389         }
554 389         }
555 389         }
556 389         }
557 389         }
558 389         }
559 389         }
560 389         }
561 389         }
562 389         }
563 389         }
564 389         }
565 389         }
566 389         }
567 389         }
568 389         }
569 389         }
570 389         }
571 389         }
572 389         }
573 389         }
574 389         }
575 389         }
576 389         }
577 389         }
578 389         }
579 389         }
580 389         }
581 389         }
582 389         }
583 389         }
584 389         }
585 389         }
586 389         }
587 389         }
588 389         }
589 389         }
590 389         }
591 389         }
592 389         }
593 389         }
594 389         }
595 389         }
596 389         }
597 389         }
598 389         }
599 389         }
600 389         }
601 389         }
602 389         }
603 389         }
604 389         }
605 389         }
606 389         }
607 389         }
608 389         }
609 389         }
610 389         }
611 389         }
612 389         }
613 389         }
614 389         }
615 389         }
616 389         }
617 389         }
618 389         }
619 389         }
620 389         }
621 389         }
622 389         }
623 389         }
624 389         }
625 389         }
626 389         }
627 389         }
628 389         }
629 389         }
630 389         }
631 389         }
632 389         }
633 389         }
634 389         }
635 389         }
636 389         }
637 389         }
638 389         }
639 389         }
640 389         }
641 389         }
642 389         }
643 389         }
644 389         }
645 389         }
646 389         }
647 389         }
648 389         }
649 389         }
650 389         }
651 389         }
652 389         }
653 389         }
654 389         }
655 389         }
656 389         }
657 389         }
658 389         }
659 389         }
660 389         }
661 389         }
662 389         }
663 389         }
664 389         }
665 389         }
666 389         }
667 389         }
668 389         }
669 389         }
670 389         }
671 389         }
672 389         }
673 389         }
674 389         }
675 389         }
676 389         }
677 389         }
678 389         }
679 389         }
680 389         }
681 389         }
682 389         }
683 389         }
684 389         }
685 389         }
686 389         }
687 389         }
688 389         }
689 389         }
690 389         }
691 389         }
692 389         }
693 389         }
694 389         }
695 389         }
696 389         }
697 389         }
698 389         }
699 389         }
700 389         }
701 389         }
702 389         }
703 389         }
704 389         }
705 389         }
706 389         }
707 389         }
708 389         }
709 389         }
710 389         }
711 389         }
712 389         }
713 389         }
714 389         }
715 389         }
716 389         }
717 389         }
718 389         }
719 389         }
720 389         }
721 389         }
722 389         }
723 389         }
724 389         }
725 389         }
726 389         }
727 389         }
728 389         }
729 389         }
730 389         }
731 389         }
732 389         }
733 389         }
734 389         }
735 389         }
736 389         }
737 389         }
738 389         }
739 389         }
740 389         }
741 389         }
742 389         }
743 389         }
744 389         }
745 389         }
746 389         }
747 389         }
748 389         }
749 389         }
750 389         }
751 389         }
752 389         }
753 389         }
754 389         }
755 389         }
756 389         }
757 389         }
758 389         }
759 389         }
760 389         }
761 389         }
762 389         }
763 389         }
764 389         }
765 389         }
766 389         }
767 389         }
768 389         }
769 389         }
770 389         }
771 389         }
772 389         }
773 389         }
774 389         }
775 389         }
776 389         }
777 389         }
778 389         }
779 389         }
780 389         }
781 389         }
782 389         }
783 389         }
784 389         }
785 389         }
786 389         }
787 389         }
788 389         }
789 389         }
790 389         }
791 389         }
792 389         }
793 389         }
794 389         }
795 389         }
796 389         }
797 389         }
798 389         }
799 389         }
800 389         }
801 389         }
802 389         }
803 389         }
804 389         }
805 389         }
806 389         }
807 389         }
808 389         }
809 389         }
810 389         }
811 389         }
812 389         }
813 389         }
814 389         }
815 389         }
816 389         }
817 389         }
818 389         }
819 389         }
820 389         }
821 389         }
822 389         }
823 389         }
824 389         }
825 389         }
826 389         }
827 389         }
828 389         }
829 389         }
830 389         }
831 389         }
832 389         }
833 389         }
834 389         }
835 389         }
836 389         }
837 389         }
838 389         }
839 389         }
840 389         }
841 389         }
842 389         }
843 389         }
844 389         }
845 389         }
846 389         }
847 389         }
848 389         }
849 389         }
850 389         }
851 389         }
852 389         }
853 389         }
854 389         }
855 389         }
856 389         }
857 389         }
858 389         }
859 389         }
860 389         }
861 389         }
862 389         }
863 389         }
864 389         }
865 389         }
866 389         }
867 389         }
868 389         }
869 389         }
870 389         }
871 389         }
872 389         }
873 389         }
874 389         }
875 389         }
876 389         }
877 389         }
878 389         }
879 389         }
880 389         }
881 389         }
882 389         }
883 389         }
884 389         }
885 389         }
886 389         }
887 389         }
888 389         }
889 389         }
890 389         }
891 389         }
892 389         }
893 389         }
894 389         }
895 389         }
896 389         }
897 389         }
898 389         }
899 389         }
900 389         }
901 389         }
902 389         }
903 389         }
904 389         }
905 389         }
906 389         }
907 389         }
908 389         }
909 389         }
910 389         }
911 389         }
912 389         }
913 389         }
914 389         }
915 389         }
916 389         }
917 389         }
918 389         }
919 389         }
920 389         }
921 389         }
922 389         }
923 389         }
924 389         }
925 389         }
926 389         }
927 389         }
928 389         }
929 389         }
930 389         }
931 389         }
932 389         }
933 389         }
934 389         }
935 389         }
936 389         }
937 389         }
938 389         }
939 389         }
940 389         }
941 389         }
942 389         }
943 389         }
944 389         }
945 389         }
946 389         }
947 389         }
948 389         }
949 389         }
950 389         }
951 389         }
952 389         }
953 389         }
954 389         }
955 389         }
956 389         }
957 389         }
958 389         }
959 389         }
960 389         }
961 389         }
962 389         }
963 389         }
964 389         }
965 389         }
966 389         }
967 389         }
968 389         }
969 389         }
970 389         }
971 389         }
972 389         }
973 389         }
974 389         }
975 389         }
976 389         }
977 389         }
978 389         }
979 389         }
980 389         }
981 389         }
982 389         }
983 389         }
984 389         }
985 389         }
986 389         }
987 389         }
988 389         }
989 389         }
990 389         }
991 389         }
992 389         }
993 389         }
994 389         }
995 389         }
996 389         }
997 389         }
998 389         }
999 389         }
1000 389         }
```

## Lucky 13 timing attack

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

# Constant-time code is hard to write

```

384 384     SSL_RECORD *rr;
385 385     unsigned int mac_size;
386 386     unsigned char
387 387     int decryption
388 388     int rrr = &(s->s3->
389 389     00 -417,13 +418,10 00
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     /* Pe
428 428     +
429 429     #ifdef TLS_DEBUG
430 430     00 -453,7 +451,7 00 p
453 451
454 452
455 453     #else
456 456     +
457 455     #endif
458 456     /* ch
459 457     00 -464,17 +462,25 00
464 462
465 463
466 464     #else
467 467     +
468 465     #endif
469 468     rrr->1
470 469     1s->
471 470     if (1
472 471
473 471
474 474     +
475 473
476 474
477 475
478 476     if (decryption
479 477     {
480 478     /* de
481 479     rrr->1
482 480     s->pad
483 481     goto
484 482     +
485 484     /* r->length
486 485     if (s->xepand
487 486     {
755 - EVP_DigestUpdate(&md_ctx,md,2);
756 - EVP_DigestUpdate(&md_ctx,rec->input,rec->length);
757 - EVP_DigestFinal
237 - unsigned char mac[SHA_DIGEST_LENGTH];
246 + union { unsigned int u[SHA_DIGEST_LENGTH/sizeof(unsigned int)];
247 + unsigned char c[SHA_DIGEST_LENGTH]; } mac;
238 248
239 249 /* decrypt HMAC(padding at once */
240 250 aesni_cbc_encrypt(in,out,len,
241 251 &key->ks,ctx->iv,0);
242 252
243 253 if (plen) { /* "TLS" mode of operation */
244 254 /* Figure out payload length */
245 255 if (len<(size_t)(out[len-1]+1+SHA_DIGEST_LENGTH))
246 256 return 0;
247 257 {
248 258 /* Th
254 + * t3
255 + * da
256 + * t3
257 + * t3
258 + * t3
259 + * t3
259 260 if ((key->aux.tls_aad[plen-4]<8)(key->aux.tls_aad[plen-3])
260 261 == TLS1_1_VERSION) {
261 262 len -= AES_BLOCK_SIZE;
262 263 == TLS1_1_VERSION)
263 264 iv = AES_BLOCK_SIZE;
264 264 }
265 265 key->aux.tls_aad[plen-2] = len>>8;
266 266 key->aux.tls_aad[plen-1] = len;
267 267 if (len<(1+SHA_DIGEST_LENGTH+1))
268 268 return 0;
269 269 /* omit explicit iv */
270 270 out += iv;
271 271 len -= iv;
272 272 /* Figure out payload length */
273 273 pad = out[len-1];
274 274 maxpad = len-(SHA_DIGEST_LENGTH+1);
275 275 maxpad |= (255-maxpad)>>(sizeof(maxpad)*8-8);
276 276 maxpad &= 255;
277 277
278 278 inp_len = len - (SHA_DIGEST_LENGTH+pad+1);
279 279 mask = (0-((inp_len-len)>>(sizeof(inp_len)*8-1)));
280 280 inp_len &= mask;
281 281 ret &= (int)mask;
282 282
283 283 /* calculate HMAC and verify it */
284 284 key->aux.tls_aad[plen-2] = inp_len>>8;
285 285 key->aux.tls_aad[plen-1] = inp_len;
286 286
287 287 /* calculate HMAC */
288 288 key->md = key->head;
289 289 SHA1_Update(&key->md,key->aux.tls_aad,plen);
290 290 SHA1_Update(&key->md,out+iv,len);
291 291 SHA1_Final(&mac,&key->md);
292 292
293 293 #if 1
294 294 len -= SHA_DIGEST_LENGTH; /* amend mac */
295 295 if (len==(256+SHA_CBLOCK)) {
296 296 j = (len-(256+SHA_CBLOCK))&(0-SHA_CBLOCK);
297 297 j += SHA_CBLOCK-key->md.num;
298 298 }

```

Further refinements  
 Removing all measurable  
 timing differences

# Goal: Write readable code

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```

```
for (i = 0; i < n; i++) {  
    d |= x[i] ^ y[i];  
}  
return (1 & ((d - 1) >> 8)) - 1;
```



# Goal: Write readable code


```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```



```
for (i = 0; i < n; i++) {  
    d |= x[i] ^ y[i];  
}  
return (1 & ((d - 1) >> 8)) - 1;
```

# Goal: Write readable code


```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```

**FaCT**  


```
for (i = 0; i < n; i++) {  
    d |= x[i] ^ y[i];  
}  
return (1 & ((d - 1) >> 8)) - 1;
```

# Goal: Write readable code

```
for (i = 0; i < n; i++) {  
    if (x[i] != y[i])  
        return -1;  
}  
return 0;
```

**FaCT** 

```
for (i = 0; i < n; i++) {  
    d |= x[i] ^ y[i];  
}  
return (1 & ((d - 1) >> 8)) - 1;
```

Transforms **readable code into constant-time code**

# Transforming to constant-time

- What to transform?
- How to transform?
- What *not* to transform?
- Evaluation

# Transforming to constant-time


- **What to transform?**
- How to transform?
- What *not* to transform?
- Evaluation

# Transform everything?

```
if (secret) {  
    x = 19;  
}
```

# Transform everything?

```
if (secret) {  
    x = 19;  
}
```



```
x = -secret & 19 | (secret-1) & x;
```

# Transform everything?

```
if (secret) {  
    x = 19;  
}
```



Slower but necessary

```
x = -secret & 19 | (secret-1) & x;
```



# Transform everything?

```
if (secret) {  
  x = 19;  
}
```



Slower but necessary

$x = -\text{secret} \& 19 \mid (\text{secret}-1) \& x;$

```
if (public) {  
  y = 42;  
}
```



$y = -\text{public} \& 42 \mid (\text{public}-1) \& y;$

# Transform everything?

```
if (secret) {  
  x = 19;  
}
```



Slower but necessary

$x = -\text{secret} \& 19 \mid (\text{secret}-1) \& x;$

```
if (public) {  
  y = 42;  
}
```



Slower and *unnecessary!*

$y = -\text{public} \& 42 \mid (\text{public}-1) \& y;$

# Transform everything?

```
if (secret) {  
  x = 19;  
}
```



Slower but necessary

$x = -\text{secret} \& 19 \mid (\text{secret}-1) \& x;$

```
if (public) {  
  y = 42;  
}
```



Slower and *unnecessary!*

$y = -\text{public} \& 42 \mid (\text{public}-1) \& y;$

Only transform if code leaks **secret values**

# Explicit secrecy in the type system

```
secret uint32 decrypt(  
    secret uint32 key,  
    public uint32 msg) {  
  
    if (key > 40) {  
        ...  
    }  
  
    ...  
}
```

# Explicit secrecy in the type system

```
secret uint32 decrypt(  
    secret uint32 key,  
    public uint32 msg) {  
  
    if (key > 40) {  
        ...  
    }  
  
    ...  
}
```

# Explicit secrecy in the type system

```
secret uint32 decrypt(  
    secret uint32 key,  
    public uint32 msg) {
```

```
    if (key > 40) {  
        ...  
    }
```

We can **detect secret leakage!**

```
    ...
```

```
}
```

# Type system detects leaks via...

- Conditional branches
- Early termination
- Function side effects
- Memory access patterns
- Direct assignment
- ...

# Type system detects leaks via...

- Conditional branches
- Early termination
- Function side effects
- Memory access patterns
- Direct assignment
- ...

**FaCT transforms these**



# Type system detects leaks via...

- Conditional branches

- Early termination

- Function side effects

- Memory access patterns

- Direct assignment

- ...

**FaCT transforms these**

**FaCT disallows these**

# Transforming to constant-time

- **What to transform?**
- How to transform?
- What *not* to transform?
- Evaluation

# Transforming to constant-time

- What to transform?
- **How to transform?**
- What *not* to transform?
- Evaluation

# Transforming control flow

- Conditional branches
- Early termination
- Function side effects

# Transforming control flow

- **Conditional branches**
- **Early termination**
- Function side effects

# Transform secret conditionals

```
if (s) {  
    x = 40;  
} else {  
    x = 19;  
    y = x + 2;  
}
```

# Transform secret conditionals

```
if (s) {  
    x = 40;  
} else {  
    x = 19;  
    y = x + 2;  
}
```



```
x = -s & 40 | (s-1) & x;
```

# Transform secret conditionals

```
if (s) {  
    x = 40;  
} else {  
    x = 19;  
    y = x + 2;  
}
```



```
x = -s & 40 | (s-1) & x;
```



# Transform secret conditionals

```
if (s) {  
    x = 40;  
} else {  
    x = 19;  
    y = x + 2;  
}
```



```
x = -s & 40 | (s-1) & x;
```



```
x = (s-1) & 19 | -s & x;  
y = (s-1) & (x + 2) | -s & y;
```

# Transform secret conditionals

```
if (s) {  
    x = 40;  
} else {  
    x = 19;  
    y = x + 2;  
}
```



```
x = -s & 40 | (s-1) & x;
```



```
x = (s-1) & 19 | -s & x;  
y = (s-1) & (x + 2) | -s & y;
```

# Secret returns are conditionals too

```
if (s) {  
    return 40;  
}
```

# Secret returns are conditionals too


```
if (s) {  
    return 40;  
}
```




```
if (s) {  
    if (!done) {  
        rval = 40;  
        done = true;  
    }  
}  
  
:  
  
return rval;
```

# Secret returns are conditionals too

```
if (s) {  
  return 40;  
}  
  
if (s) {  
  if (!done) {  
    rval = 40;  
    done = true;  
  }  
}
```



# Secret returns are conditionals too

```
if (s) {  
  return 40;  
} 

```
if (s) {  
  if (!done) {  
    rval = 40;  
    done = true;  
  }  
}
```


```

# Secret returns are conditionals too

```
if (s) {  
  return 40;  
}  
→  
if (s) {  
  if (!done) {  
    rval = 40;  
    done = true;  
  }  
}  
→  
rval = (-s & (done-1)) & 40 | ...  
done = (-s & (done-1)) & true | ...
```

# Transforming to constant-time

- What to transform?
- **How to transform?**
- What *not* to transform?
- Evaluation



# Transforming to constant-time

- What to transform?
- How to transform?
- **What *not* to transform?**
- Evaluation

# Not all transformations are good

- May produce **inefficient** code
- May produce **unsafe** code

# Not all transformations are good

- May produce **inefficient** code
- May produce **unsafe** code

**Type system rejects** such programs

# Inefficient transformations

```
x = buffer[secret_index];
```

# Inefficient transformations

```
x = buffer[secret_index];
```



```
for (uint32 i from 0 to len buffer) {  
  if (i == secret_index) {  
    x = buffer[i];  
  }  
}
```

# Inefficient transformations

**O(1)**

```
x = buffer[secret_index];
```



**O(n)**

```
for (uint32 i from 0 to len buffer) {  
  if (i == secret_index) {  
    x = buffer[i];  
  }  
}
```

# Inefficient transformations

**O(1)**

```
x = buffer[secret_index];
```



**O(n)**

```
for (uint32_t i from 0 to buffer) {  
  if (i == secret_index) {  
    x = buffer[i];  
  }  
}
```



# Inefficient transformations

**$O(1)$**

```
x = buffer[secret_index];
```



**$O(n)$**

```
for (uint32_t i from 0 to buffer) {  
  if (i == secret_index) {  
    x = buffer[i];  
  }  
}
```



**Reject** if transformation is **inefficient**



# Unsafe transformations

```
if (j < secret_len) {  
    x = arr[j];  
}
```

# Unsafe transformations

```
if (j < secret_len) {  
    x = arr[j];  
}
```



```
x = -(j < secret_len) & arr[j]  
    | ((j < secret_len)-1) & x;
```

# Unsafe transformations

```
if (j < secret_len) {  
    x = arr[j];  
}
```




```
x = -(j < secret_len) & arr[j]  
    | ((j < secret_len)-1) & x;
```




# Unsafe transformations

```
if (j < secret_len) {  
    x = arr[j];  
}
```




```
x = -(j < secret_len) & arr[j]  
    | ((j < secret_len)-1) & x;
```




What if  $j > \text{len arr}$ ?

# Unsafe transformations

```
if (j < secret_len) {  
    x = arr[j];  
}
```



```
x = -(j < secret_len) & arr[j]  
    | ((j < secret_len)-1) & x;
```



What if  $j > \text{len arr}$ ?

**Out of bounds access!**

# Type system checks safety

Check for **out-of-bounds accesses**

**Solve constraints** using Z3

Path sensitive *except secret branches*

**Reject** if transformation is **unsafe**

# Type system checks safety

Check for **out-of-bounds accesses**

**Solve constraints** using Z3

Path sensitive *except secret branches*

**Reject** if transformation is **unsafe**

# Type system checks safety

Check for **out-of-bounds accesses**

**Solve constraints** using Z3

Path sensitive *except secret branches*

**Reject** if transformation is **unsafe**



# Type system checks safety

Check for **out-of-bounds accesses**

**Solve constraints** using Z3

Path sensitive ***except secret branches***

**Reject** if transformation is **unsafe**

# Type system checks safety

Check for **out-of-bounds accesses**

**Solve constraints** using Z3

Path sensitive *except secret branches*

**Reject** if transformation is **unsafe**

# Transforming to constant-time

- What to transform?
- How to transform?
- **What *not* to transform?**
- Evaluation

# Transforming to constant-time

- What to transform?
- How to transform?
- What *not* to transform?
- **Evaluation**

# Evaluating FaCT

- Can FaCT express real code?
- Is FaCT code as fast as C?
- Is FaCT more readable than C?

# Evaluating FaCT

- **Can FaCT express real code?**
- Is FaCT code as fast as C?
- Is FaCT more readable than C?

# Porting code to FaCT

- Rewrite the whole library
- Rewrite a function (and callees)
- Rewrite a chunk of code

# Porting code to FaCT

- Rewrite the **whole library**
- Rewrite a function (and callees)
- Rewrite a chunk of code



# Porting code to FaCT

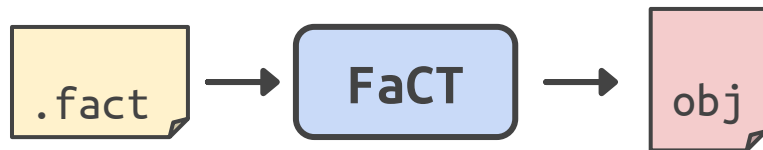
- Rewrite the whole library
- Rewrite a **function (and callees)**
- Rewrite a chunk of code

# Porting code to FaCT

- Rewrite the whole library
- Rewrite a function (and callees)
- Rewrite a **chunk of code**

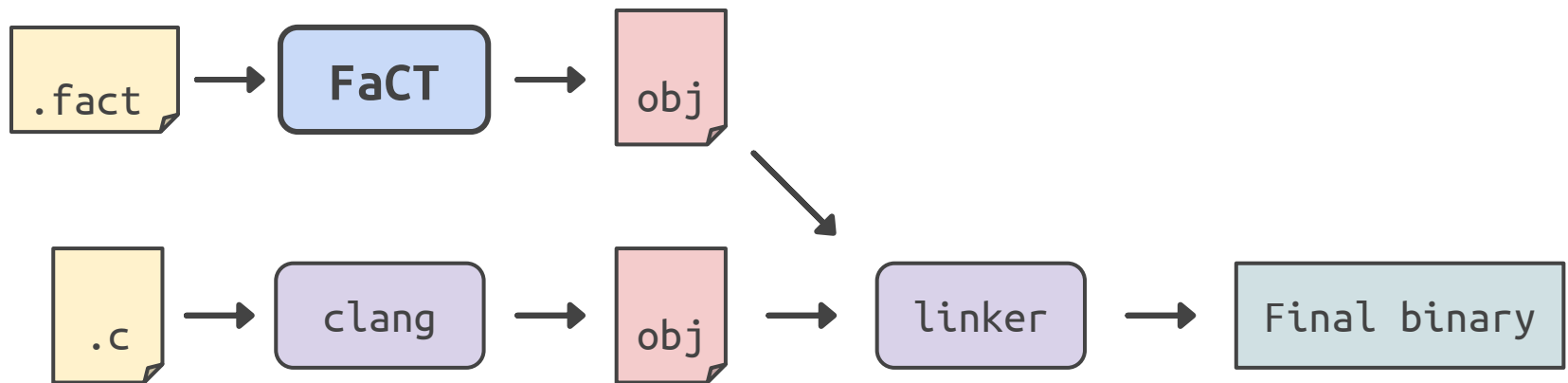
# Porting code to FaCT

- Rewrite the whole library
- Rewrite a function (and callees)
- Rewrite a chunk of code



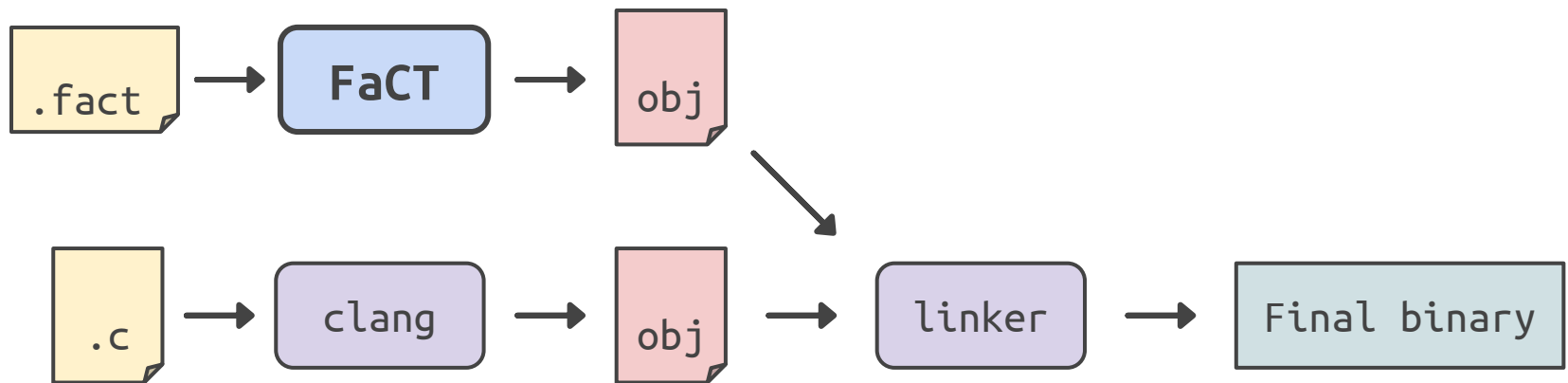
# Porting code to FaCT

- Rewrite the whole library
- Rewrite a function (and callees)
- Rewrite a chunk of code



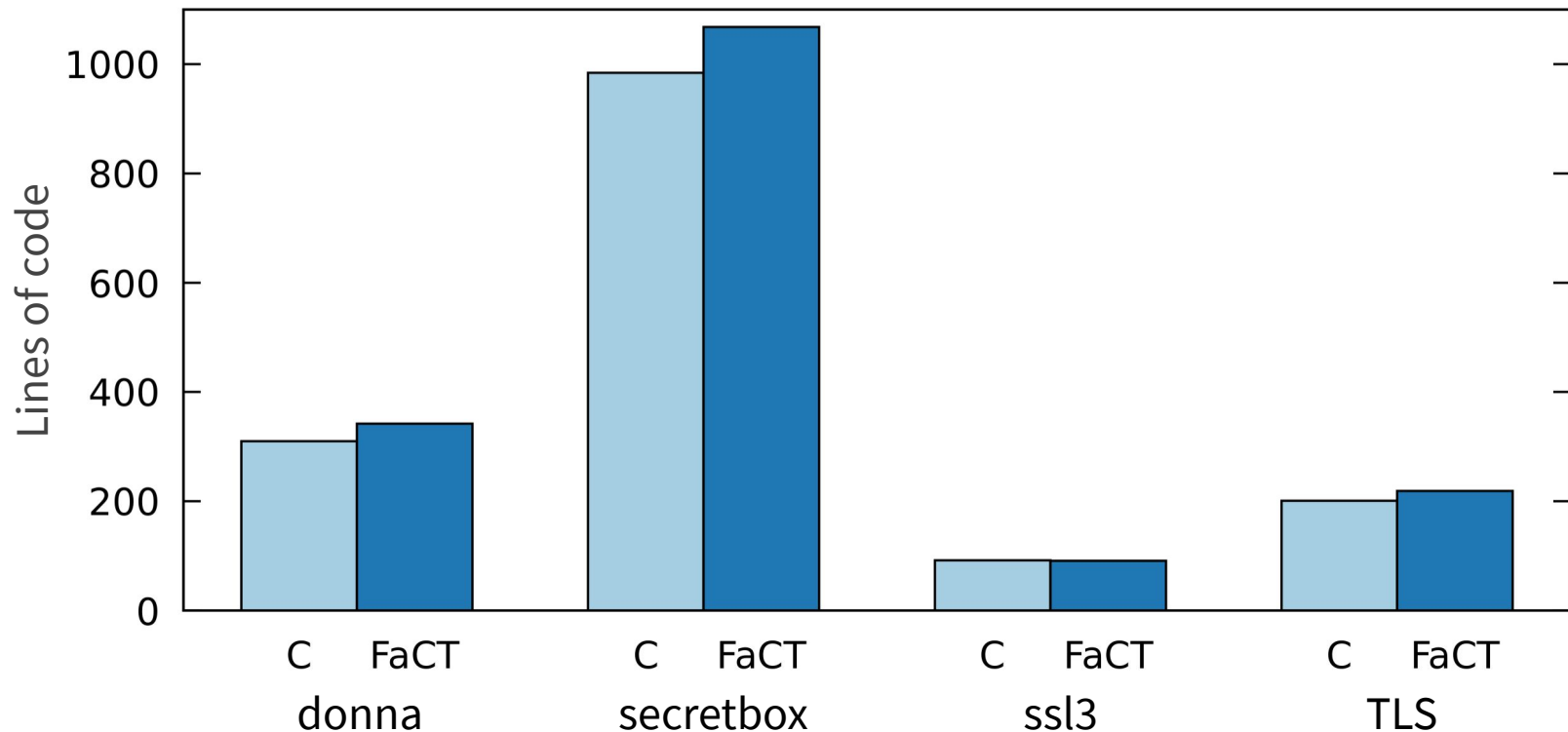
# Porting code to FaCT

- Rewrite the whole library: **donna curve25519**
- Rewrite a function (and callees): **libsodium secretbox**
- Rewrite a chunk of code: **OpenSSL ssl3/TLS record verification**



# Porting code to FaCT

- Rewrite the whole library: **donna curve25519**
- Rewrite a function (and callees): **libsodium secretbox**
- Rewrite a chunk of code: **OpenSSL ssl3/TLS record verification**



# Real code needs escape hatches

- Declassify
- Assume
- Extern

# Real code needs escape hatches

- **Declassify** secrets to public
- Assume
- Extern



# Real code needs escape hatches

- **Declassify** secrets to public

- secretbox: 

```
if (!declassify(crypto_verify(...)))  
    return false;
```

- Assume

- Extern

# Real code needs escape hatches

- **Declassify** secrets to public

- secretbox: 

```
if (!declassify(crypto_verify(...)))  
    return false;
```
- TLS: 

```
b = pmac[declassify(i)];
```

- Assume

- Extern

# Real code needs escape hatches

- **Declassify** secrets to public

- secretbox: 

```
if (!declassify(crypto_verify(...)))  
    return false;
```
- TLS: 

```
b = pmac[declassify(i)];
```

- **Assume** constraints for solver

- Extern

# Real code needs escape hatches

- **Declassify** secrets to public

- secretbox: `if (!declassify(crypto_verify(...)))  
return false;`
- TLS: `b = pmac[declassify(i)];`

- **Assume** constraints for solver

- Function preconditions

- Extern

# Real code needs escape hatches

- **Declassify** secrets to public

- secretbox: 

```
if (!declassify(crypto_verify(...)))  
    return false;
```
- TLS: 

```
b = pmac[declassify(i)];
```

- **Assume** constraints for solver

- Function preconditions
- Invariants for mutable variables

- **Extern**

# Real code needs escape hatches

- **Declassify** secrets to public

- secretbox: 

```
if (!declassify(crypto_verify(...)))  
    return false;
```
- TLS: 

```
b = pmac[declassify(i)];
```

- **Assume** constraints for solver

- Function preconditions
- Invariants for mutable variables

- **Extern** function declarations

# Real code needs escape hatches

- **Declassify** secrets to public

- secretbox: 

```
if (!declassify(crypto_verify(...)))  
    return false;
```
- TLS: 

```
b = pmac[declassify(i)];
```

- **Assume** constraints for solver

- Function preconditions
- Invariants for mutable variables

- **Extern** function declarations

- OpenSSL: AES + SHA1 implementations

# Evaluating FaCT

- Can FaCT express real code?
- **Is FaCT code as fast as C?**
- Is FaCT more readable than C?



# Performance vs. C

- Optimized with same optimization flags
- Empirically tested to be constant-time

# Performance vs. C

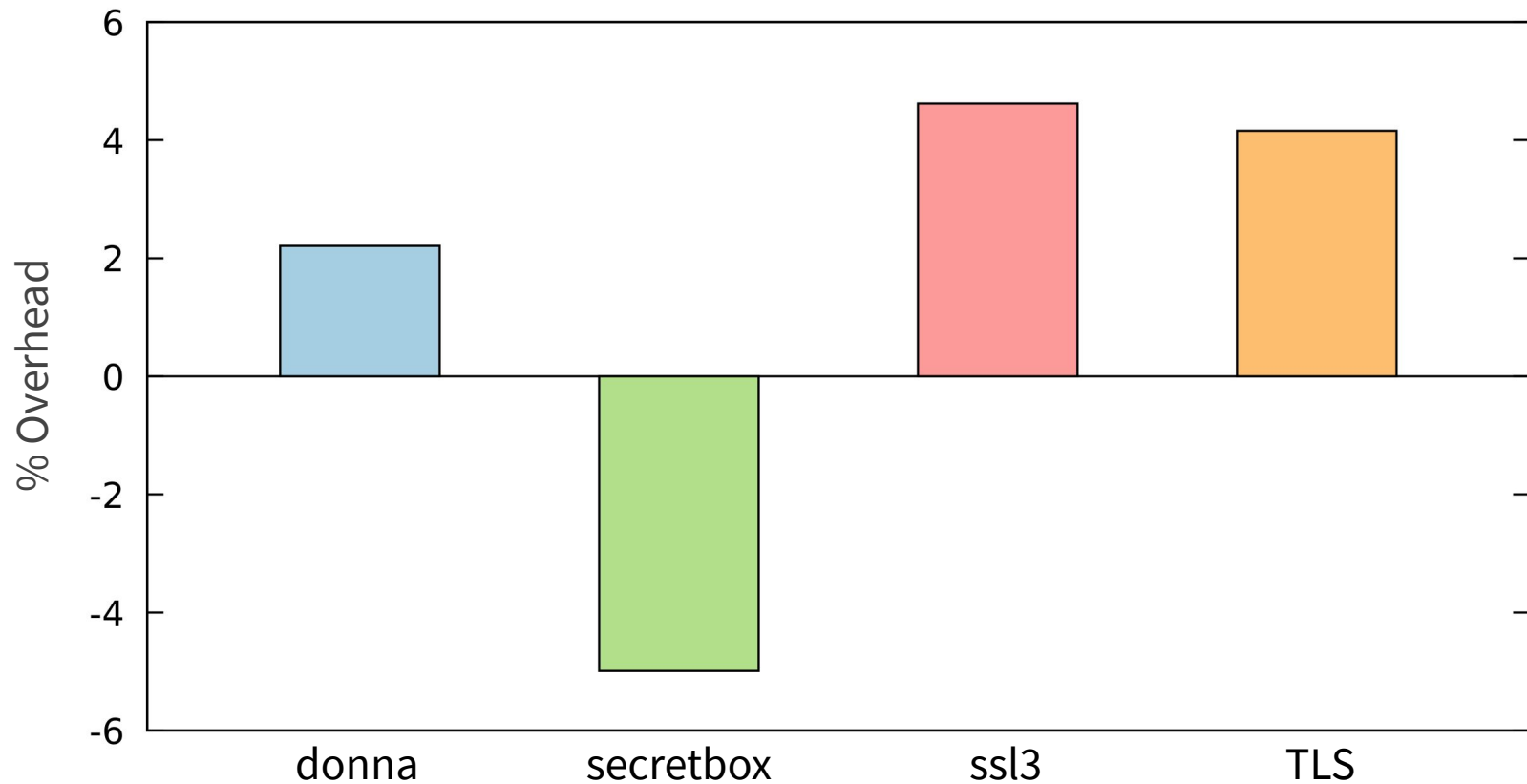
- Optimized with **same optimization flags**
- Empirically tested to be constant-time

# Performance vs. C

- Optimized with **same optimization flags**
- **Empirically tested** to be constant-time

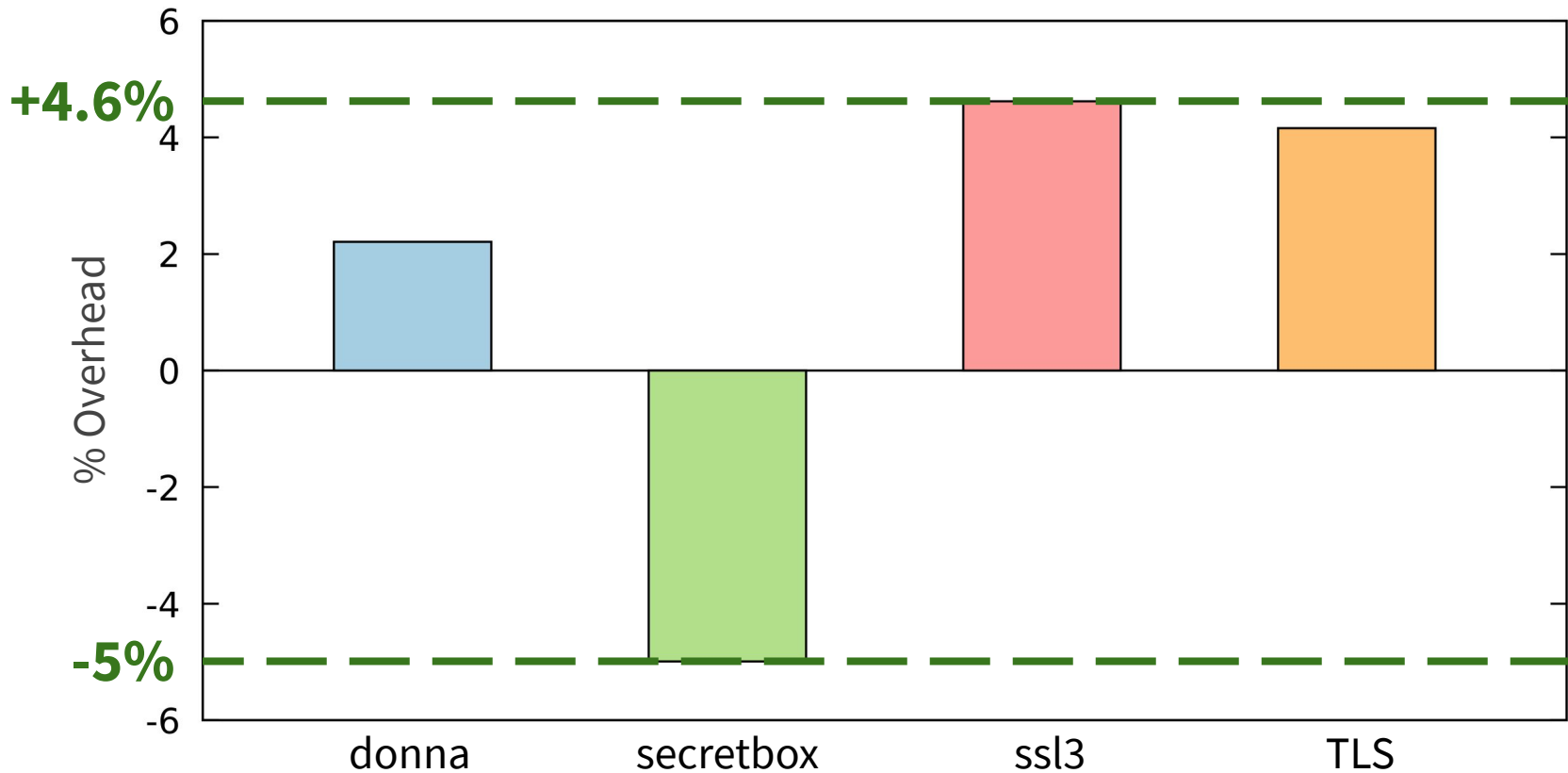
# Performance vs. C

- Optimized with **same optimization flags**
- **Empirically tested** to be constant-time



# Performance vs. C

- Optimized with **same optimization flags**
- **Empirically tested** to be constant-time



# Evaluating FaCT

- Can FaCT express real code?
- Is FaCT code as fast as C?
- **Is FaCT more readable than C?**

# User study: FaCT vs. C

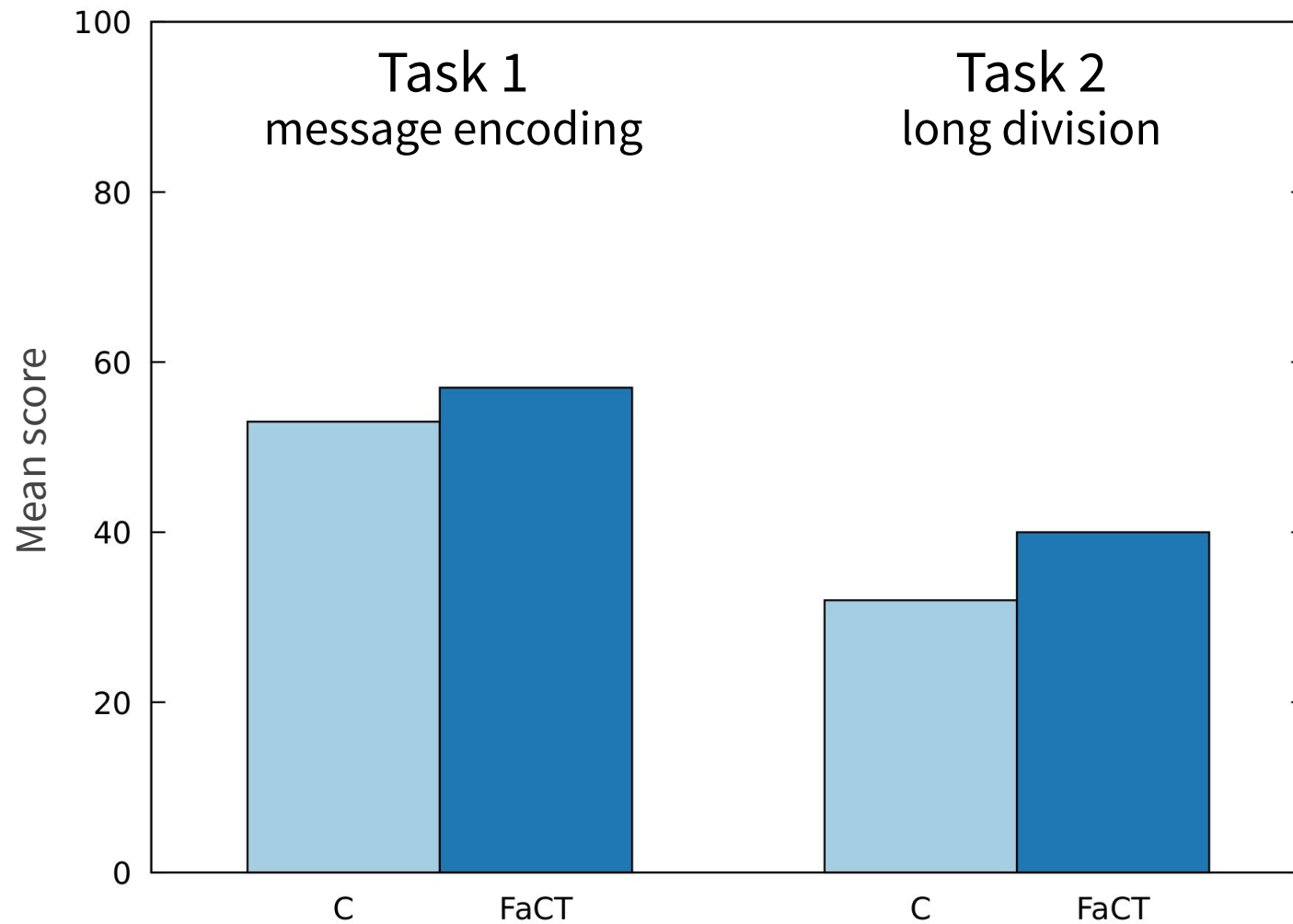
- 77 undergraduates
- Understanding constant-time code
- Writing constant-time code

# User study: FaCT vs. C

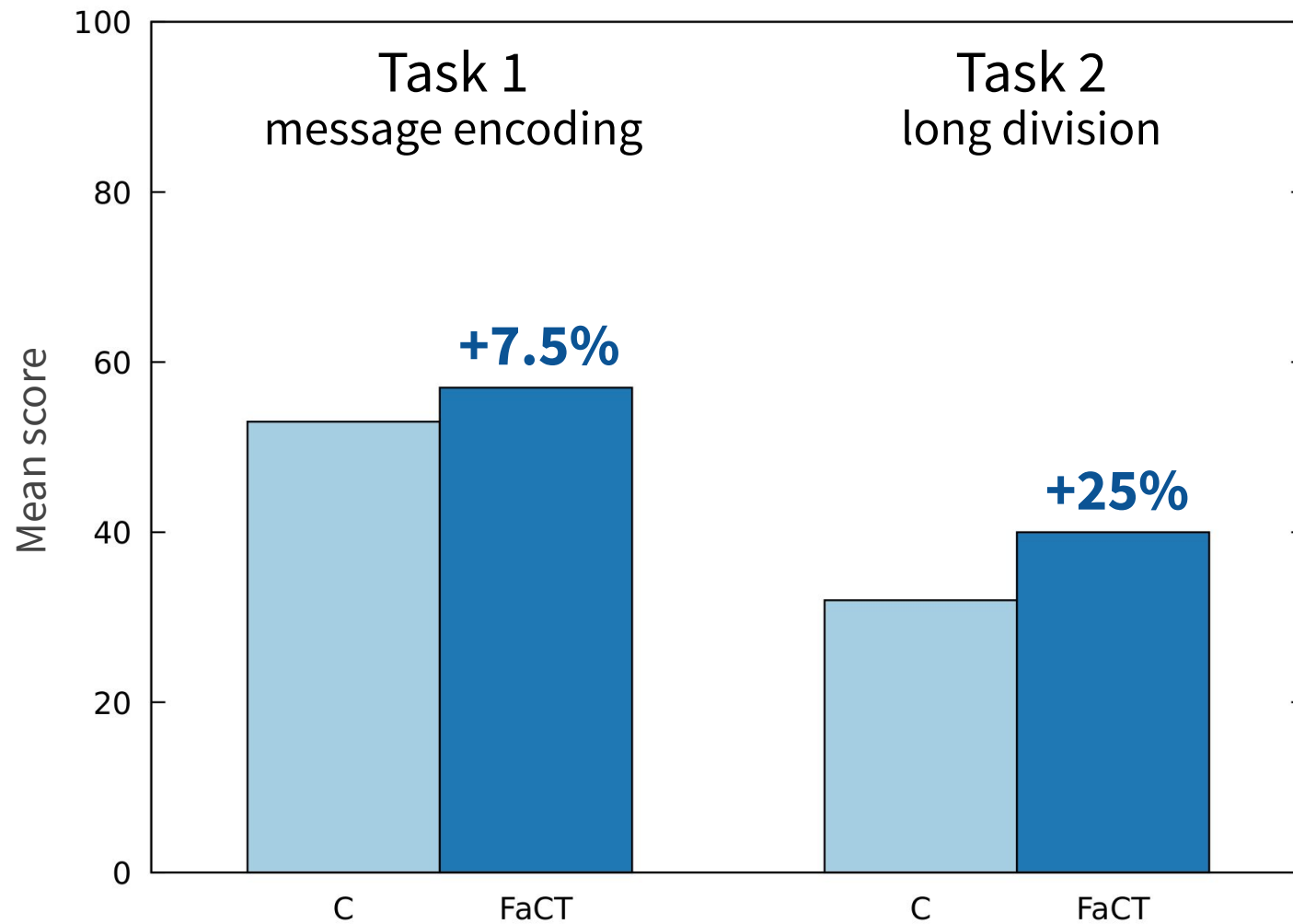
- 77 undergraduates
- **Understanding** constant-time code
- **Writing** constant-time code



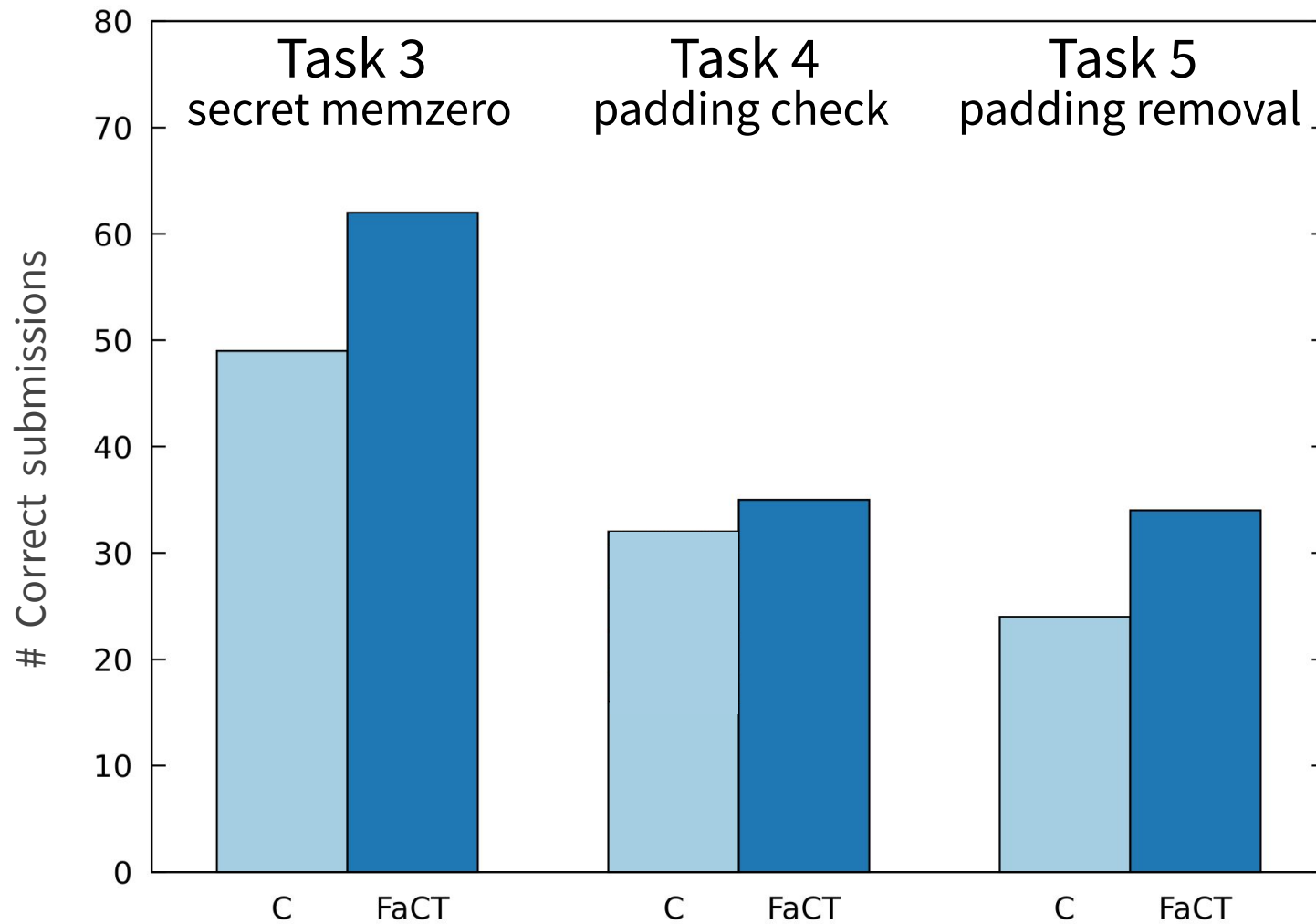
# Understanding constant-time code



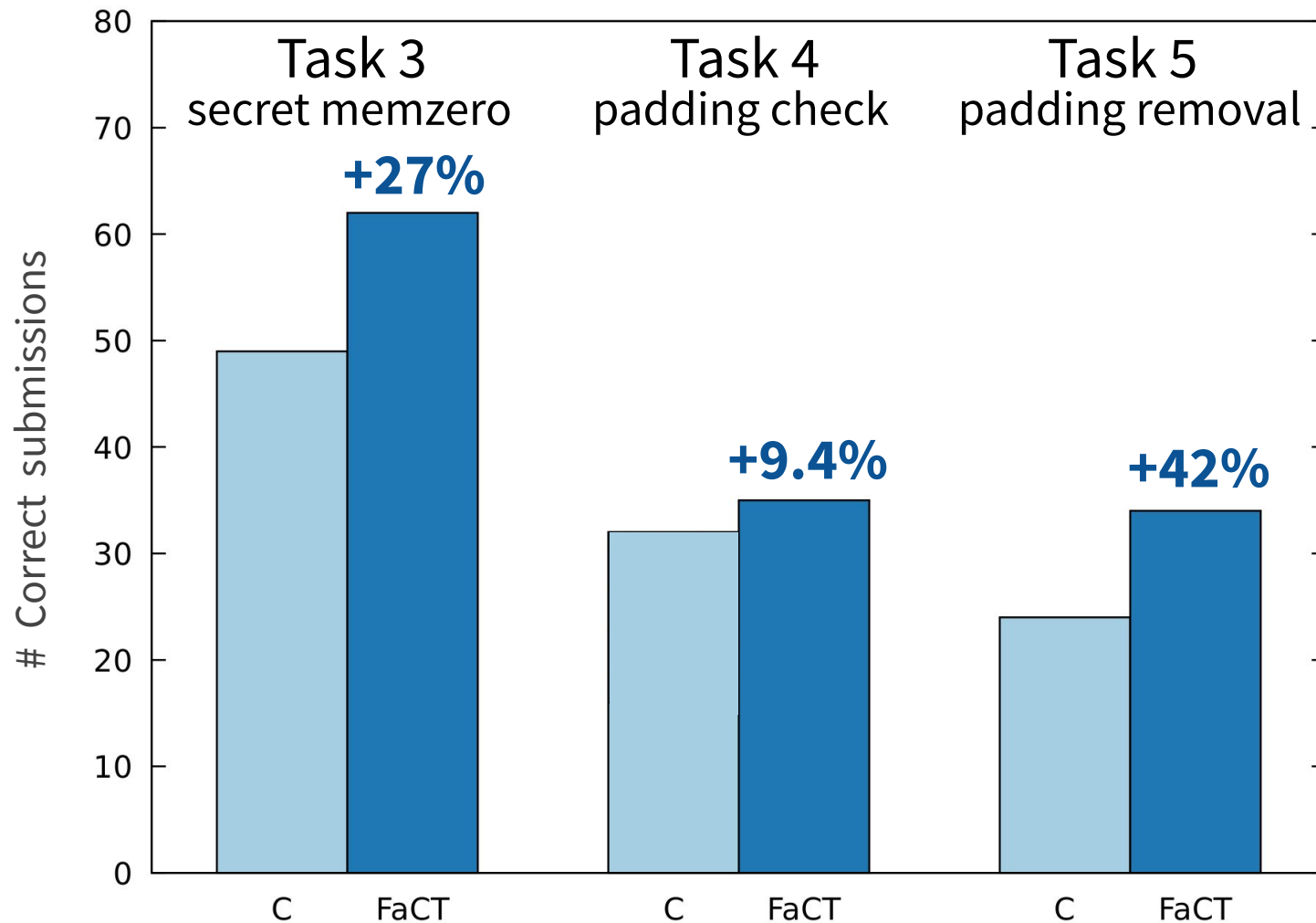
# Understanding constant-time code



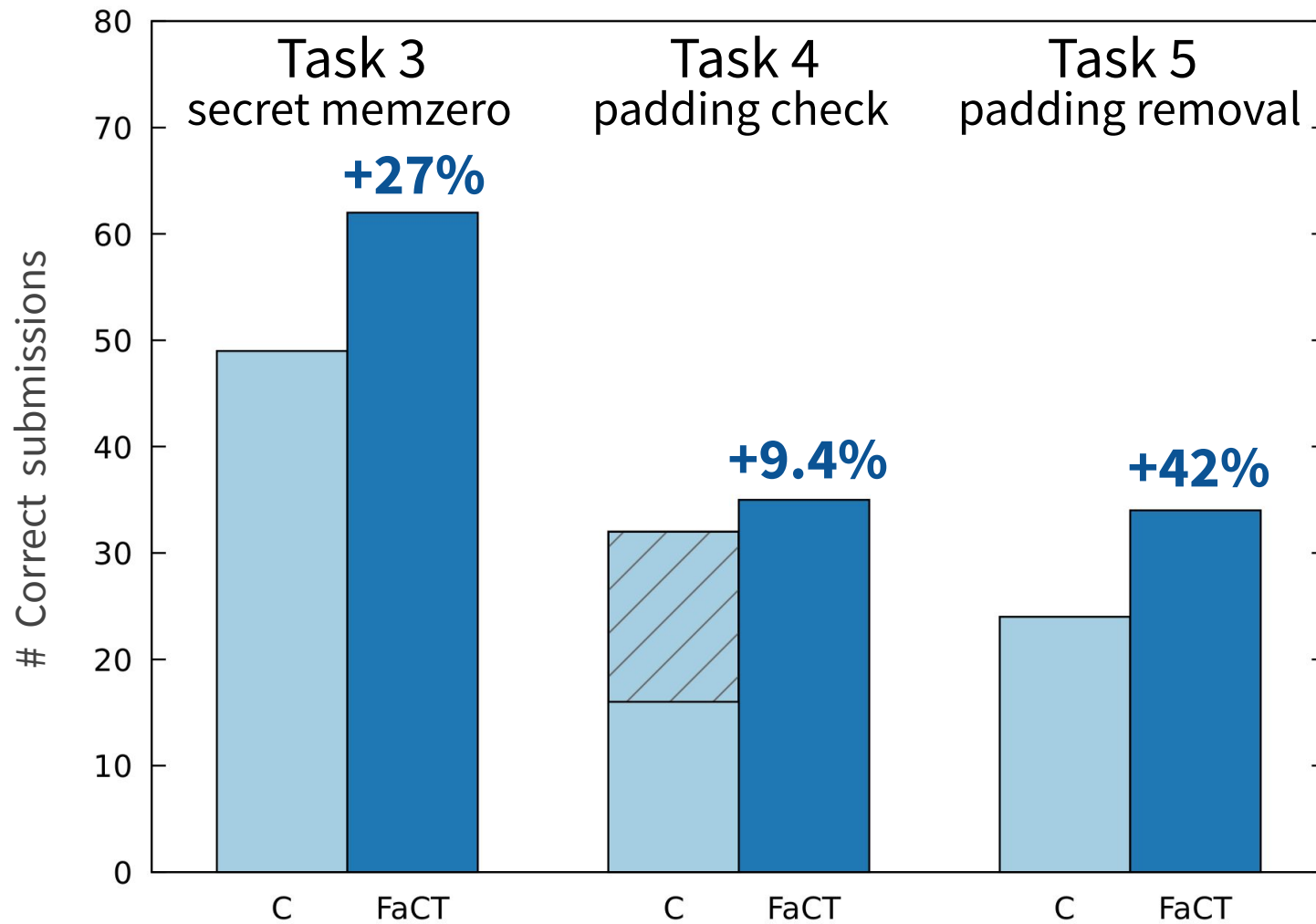
# Writing constant-time code



# Writing constant-time code



# Writing constant-time code



# Evaluating FaCT

- FaCT can express real code
- FaCT code is as fast as C
- FaCT is more readable than C

# Summary

- DSL for writing **readable** constant-time code
- **Transform secret control flow** to constant-time
- Ensure transformations **can be performed safely**

<https://fact.programming.systems>





# Comparing two buffers in FaCT

```
secret int32 crypto_verify_n(  
    secret uint8[] x,  
    secret uint8[] y) {  
  
    assume(len x == len y);  
    for (uint64 i from 0 to len x) {  
        if (x[i] != y[i]) {  
            return -1;  
        }  
    }  
    return 0;  
}
```

# Message encoding in FaCT

```
for (uint64 j from 0 to md_block_size) {
  secret mut uint8 b = 0;
  b = data[j];
  if (is_block_a) {
    if (j == c) {
      b = 0x80;
    } else if (j > c) {
      b = 0;
    }
  }
  if (is_block_b)
    if (!is_block_a) {
      b = 0;
    }
    if (j >= md_block_size - md_length_size) {
      b = length_bytes[j - (md_block_size - md_length_size)];
    }
  }
  block[j] = b;
}
```