

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.

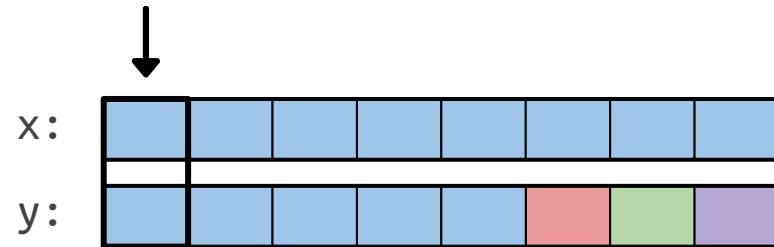
```
for (i = 0; i < n; i++) {
    if (x[i] != y[i])
        return -1;
}
return 0;
```

It compares two buffers.

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

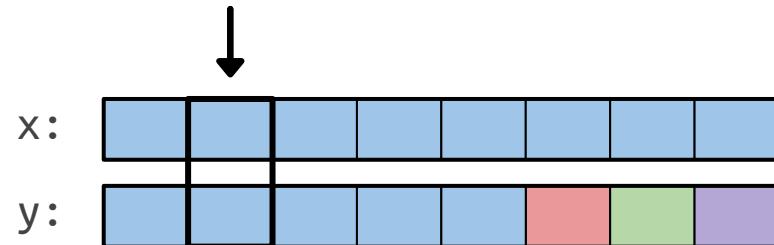
It compares two buffers.

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



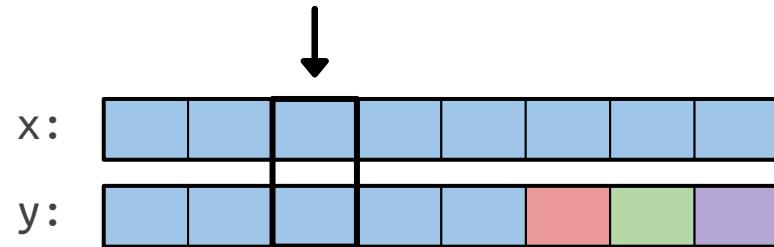
It compares two buffers.

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



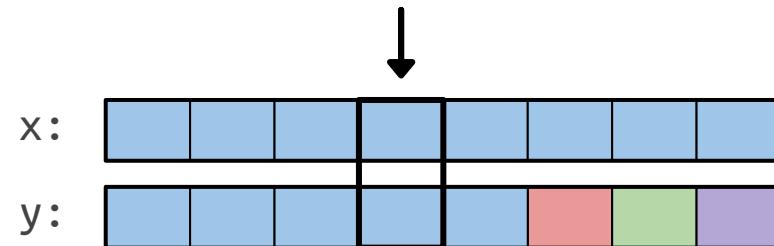
It compares two buffers.

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



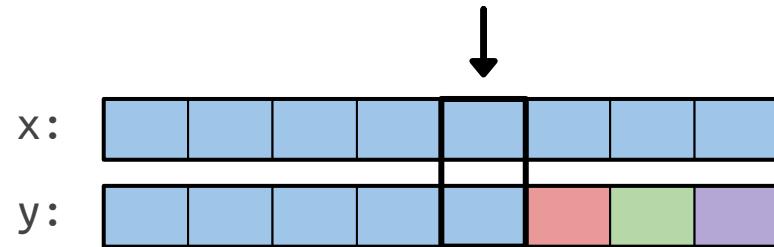
It compares two buffers.

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



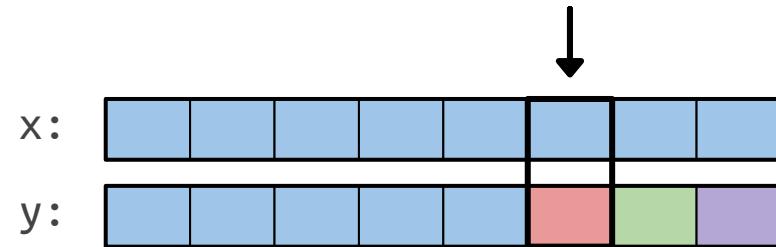
It compares two buffers.

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



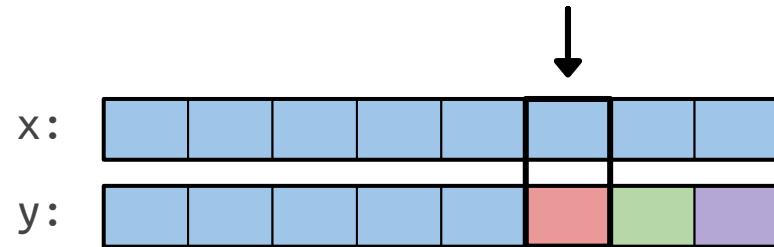
It compares two buffers.

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



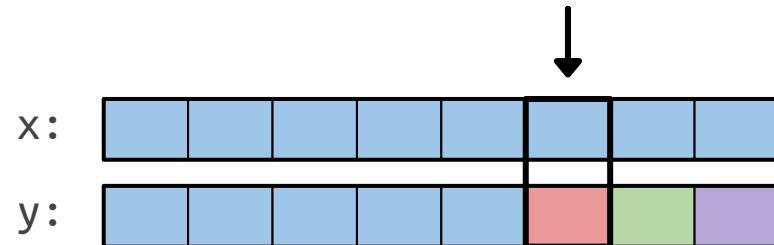
It compares two buffers.

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



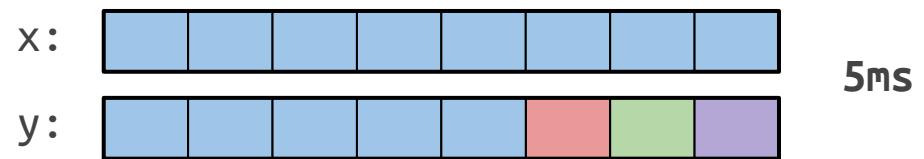
It compares two buffers.

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



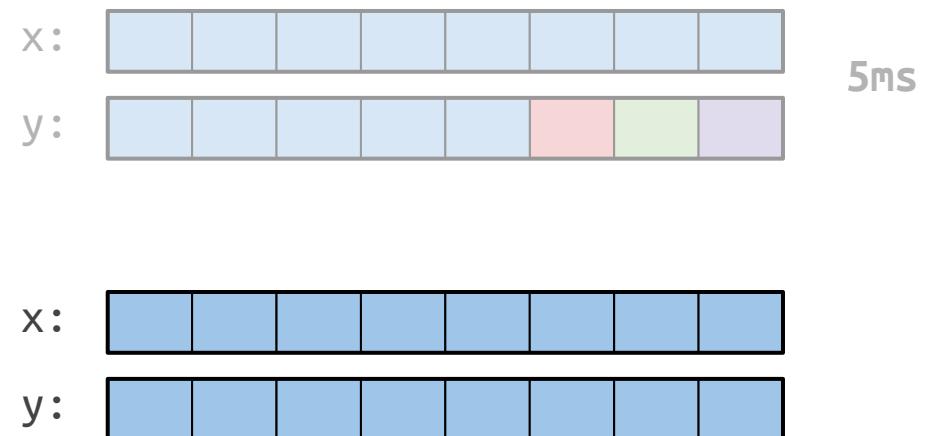
It compares two buffers.

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



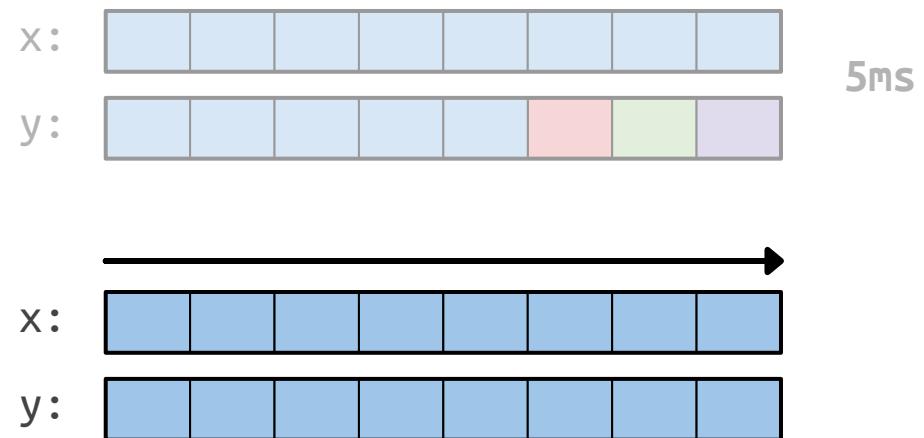
It compares two buffers.

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



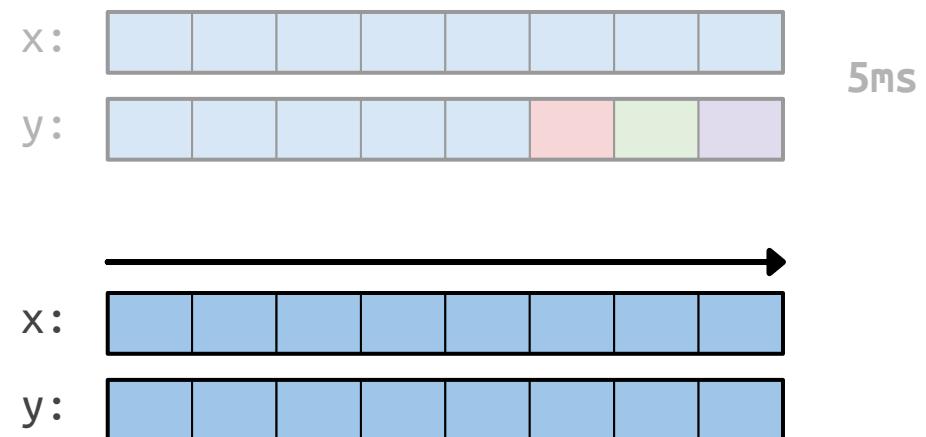
It compares two buffers.

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



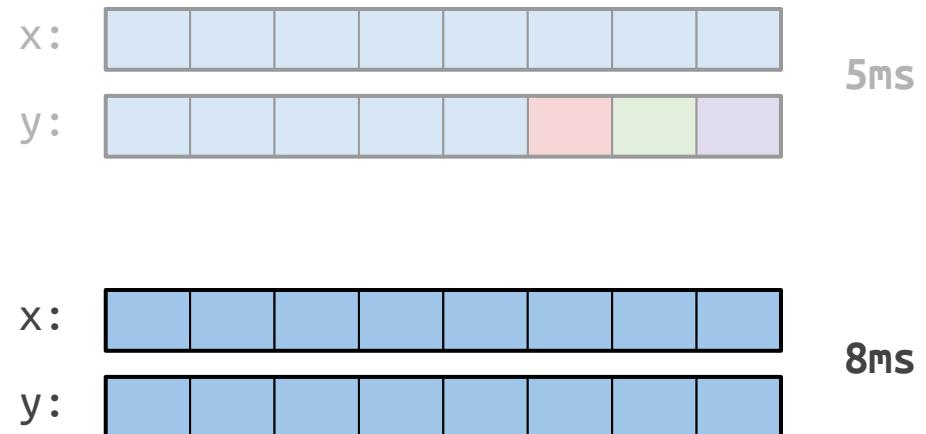
It compares two buffers.

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



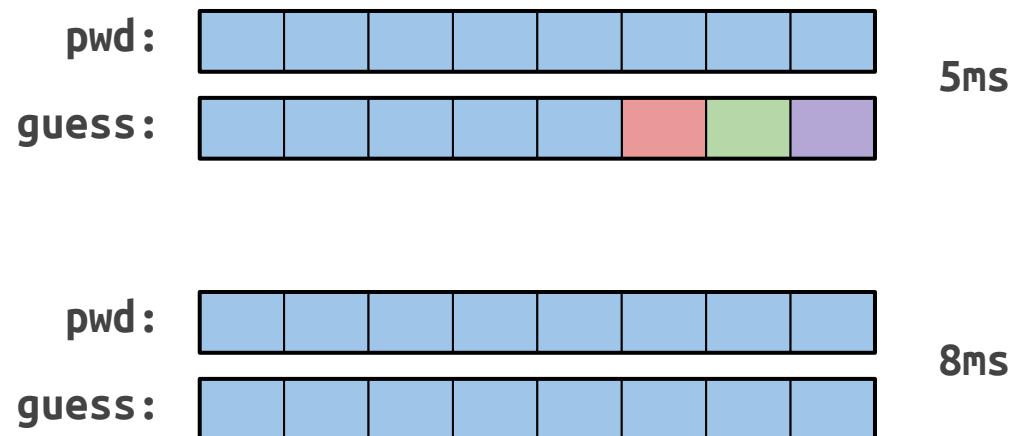
It compares two buffers.

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



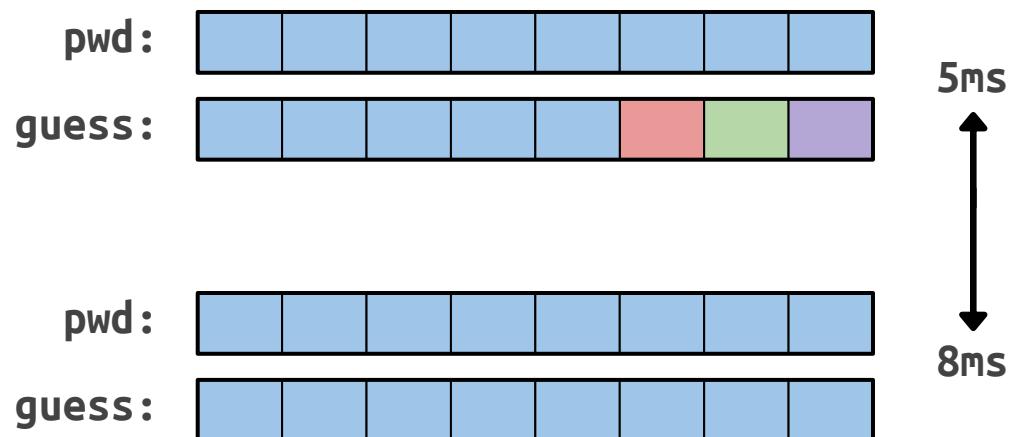
It compares two buffers.

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



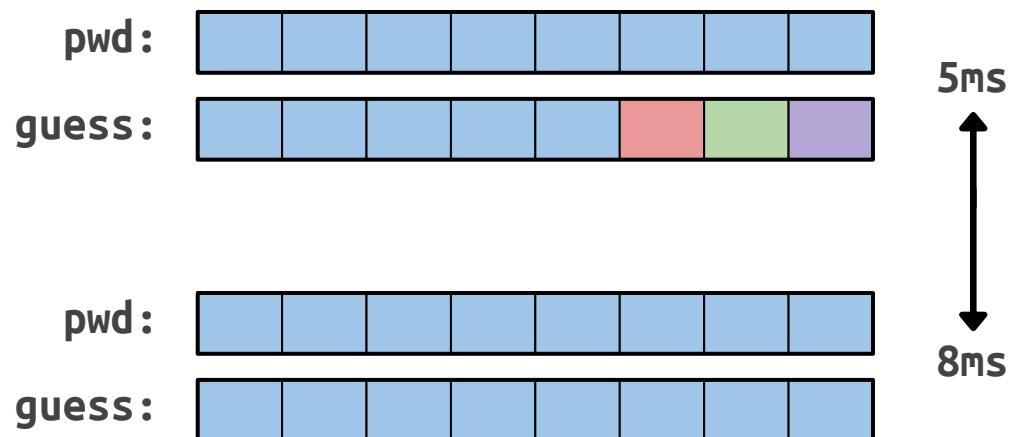
It compares two buffers.

```
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

```
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;
}
```

NOT READABLE!

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

```
383 383     SSL3_RECORD *rr;
384 384     unsigned int mac_size;
385 385     unsigned char md[EVP_MAX_MD_SIZE];
386 +     int decryption_failed_or_bad_record_mac = 0;
386 387
387 388
388 389     rr = &(s->s3->rrec);
389 390     #0 -417,13 +418,19 #0 otls1_process_record(SSL *)
417 418     enc_err = s->method->ssl3_enc->enc(s, 0);
418 419     if (enc_err <= 0)
419 420     {
420     /* decryption failed, silently discard message */
421     if (enc_err < 0)
422     {
423         rr->length = 0;
424         s->packet_length = 0;
425     }
426     goto err;
421 +     /* To minimize information leaked via timing, we will always
422 +     * perform all computations before discarding the message.
423 +     */
424 +     decryption_failed_or_bad_record_mac = 1;
427 425
428 426
429 427     #ifdef TLS_DEBUG
429 428     #0 -453,7 +451,7 #0 printf("\n");
430 431     SSLerr(SSL_F_DTLS1_PROCESS_RECORD,SSL_R_PRE_MAC_LENGTH_TOO_LONG);
431 432     goto f_err;
454 452     #else
455 453     #endif
456     goto err;
454 +     decryption_failed_or_bad_record_mac = 1;
457 456     #endif
458 457     /* check the MAC for rr->input (it's in mac_size bytes at the tail) */
458 458     #0 -464,17 +462,25 #0 printf("\n");
464 462     SSLerr(SSL_F_DTLS1_PROCESS_RECORD,SSL_R_LENGTH_TOO_SHORT);
465 463     goto f_err;
466 464     #else
467     goto err;
465 +     decryption_failed_or_bad_record_mac = 1;
468 466     #endif
469 467     rr->length -= mac_size;
470 469     i = s->method->ssl3_enc->mac(s, md, 0);
472 470     if (i < 0 || memcmp(md, &(rr->data[rr->length]), mac_size) != 0)
473 471     {
474     goto err;
472 +     decryption_failed_or_bad_record_mac = 1;
473 474     }
475 475
476 +     if (decryption_failed_or_bad_record_mac)
477     {
478     /* decryption failed, silently discard message */
479     rr->length = 0;
480     s->packet_length = 0;
481     goto err;
482     }
483 +
478 484     /* r->length is now just compressed */
479 485     if (s->expand != NULL)
480     {
481
```

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 +    int decryption;
387 387
388 389     rr= &(s->s3->r);
389 390
417 418     enc_err = s->err;
418 419     if (enc_err <
419 420     {
420 421     /* dec */
421 422     if (decryption)
422 423     {
423 424     /* To */
424 425     /* per */
424 426     /* */
424 427     /* decry */
427 428     }
428 429     #ifdef TLS_DEBUG
429 430     00 -453.7 +451.7 00 p
430 431
453 451
454 452
455 453     #else
456 454     -
457 455     #endif
458 456
459 457     /* che */
459 460
460 461     00 -464.17 +462.25 00
461 462
465 463
466 464     #else
467 465     -
468 466     #endif
469 467
470 468     rr->r;
471 469     i=s->r;
472 470     if (1
473 471
474 472     -
475 473
476 474     }
477 475
476 476 +    if (decryption)
477 477     {
478 478     /* de */
479 479     rr->r;
480 480     s->p+=8;
481 481     goto e;
482 482     }
483 483
478 484     /* r->length */
479 485     if (s->expand)
480 486     {
481 487
482 488
483 489
484 490
485 491
486 492
487 493
488 494
489 495
490 496
491 497
492 498
493 499
494 500
495 501
496 502
497 503
498 504
499 505
500 506
501 507
502 508
503 509
504 510
505 511
506 512
507 513
508 514
509 515
510 516
511 517
512 518
513 519
514 520
515 521
516 522
517 523
518 524
519 525
520 526
521 527
522 528
523 529
524 530
525 531
526 532
527 533
528 534
529 535
530 536
531 537
532 538
533 539
534 540
535 541
536 542
537 543
538 544
539 545
540 546
541 547
542 548
543 549
544 550
545 551
546 552
547 553
548 554
549 555
550 556
551 557
552 558
553 559
554 560
555 561
556 562
557 563
558 564
559 565
560 566
561 567
562 568
563 569
564 570
565 571
566 572
567 573
568 574
569 575
570 576
571 577
572 578
573 579
574 580
575 581
576 582
577 583
578 584
579 585
580 586
581 587
582 588
583 589
584 590
585 591
586 592
587 593
588 594
589 595
590 596
591 597
592 598
593 599
594 600
595 601
596 602
597 603
598 604
599 605
590 606
591 607
592 608
593 609
594 610
595 611
596 612
597 613
598 614
599 615
590 616
591 617
592 618
593 619
594 620
595 621
596 622
597 623
598 624
599 625
590 626
591 627
592 628
593 629
594 630
595 631
596 632
597 633
598 634
599 635
590 636
591 637
592 638
593 639
594 640
595 641
596 642
597 643
598 644
599 645
590 646
591 647
592 648
593 649
594 650
595 651
596 652
597 653
598 654
599 655
590 656
591 657
592 658
593 659
594 660
595 661
596 662
597 663
598 664
599 665
590 666
591 667
592 668
593 669
594 670
595 671
596 672
597 673
598 674
599 675
590 676
591 677
592 678
593 679
594 680
595 681
596 682
597 683
598 684
599 685
590 686
591 687
592 688
593 689
594 690
595 691
596 692
597 693
598 694
599 695
590 696
591 697
592 698
593 699
594 700
595 701
596 702
597 703
598 704
599 705
590 706
591 707
592 708
593 709
594 710
595 711
596 712
597 713
598 714
599 715
590 716
591 717
592 718
593 719
594 720
595 721
596 722
597 723
598 724
599 725
590 726
591 727
592 728
593 729
594 730
595 731
596 732
597 733
598 734
599 735
590 736
591 737
592 738
593 739
594 740
595 741
596 742
597 743
598 744
599 745
590 746
591 747
592 748
593 749
594 750
595 751
596 752
597 753
598 754
599 755
590 756
591 757
592 758
593 759
594 760
595 761
596 762
597 763
598 764
599 765
590 766
591 767
592 768
593 769
594 770
595 771
596 772
597 773
598 774
599 775
590 776
591 777
592 778
593 779
594 780
595 781
596 782
597 783
598 784
599 785
590 786
591 787
592 788
593 789
594 790
595 791
596 792
597 793
598 794
599 795
590 796
591 797
592 798
593 799
594 800
595 801
596 802
597 803
598 804
599 805
590 806
591 807
592 808
593 809
594 810
595 811
596 812
597 813
598 814
599 815
590 816
591 817
592 818
593 819
594 820
595 821
596 822
597 823
598 824
599 825
590 826
591 827
592 828
593 829
594 830
595 831
596 832
597 833
598 834
599 835
590 836
591 837
592 838
593 839
594 840
595 841
596 842
597 843
598 844
599 845
590 846
591 847
592 848
593 849
594 850
595 851
596 852
597 853
598 854
599 855
590 856
591 857
592 858
593 859
594 860
595 861
596 862
597 863
598 864
599 865
590 866
591 867
592 868
593 869
594 870
595 871
596 872
597 873
598 874
599 875
590 876
591 877
592 878
593 879
594 880
595 881
596 882
597 883
598 884
599 885
590 886
591 887
592 888
593 889
594 890
595 891
596 892
597 893
598 894
599 895
590 896
591 897
592 898
593 899
594 900
595 901
596 902
597 903
598 904
599 905
590 906
591 907
592 908
593 909
594 910
595 911
596 912
597 913
598 914
599 915
590 916
591 917
592 918
593 919
594 920
595 921
596 922
597 923
598 924
599 925
590 926
591 927
592 928
593 929
594 930
595 931
596 932
597 933
598 934
599 935
590 936
591 937
592 938
593 939
594 940
595 941
596 942
597 943
598 944
599 945
590 946
591 947
592 948
593 949
594 950
595 951
596 952
597 953
598 954
599 955
590 956
591 957
592 958
593 959
594 960
595 961
596 962
597 963
598 964
599 965
590 966
591 967
592 968
593 969
594 970
595 971
596 972
597 973
598 974
599 975
590 976
591 977
592 978
593 979
594 980
595 981
596 982
597 983
598 984
599 985
590 986
591 987
592 988
593 989
594 990
595 991
596 992
597 993
598 994
599 995
590 996
591 997
592 998
593 999
594 1000
595 1001
596 1002
597 1003
598 1004
599 1005
590 1006
591 1007
592 1008
593 1009
594 1010
595 1011
596 1012
597 1013
598 1014
599 1015
590 1016
591 1017
592 1018
593 1019
594 1020
595 1021
596 1022
597 1023
598 1024
599 1025
590 1026
591 1027
592 1028
593 1029
594 1030
595 1031
596 1032
597 1033
598 1034
599 1035
590 1036
591 1037
592 1038
593 1039
594 1040
595 1041
596 1042
597 1043
598 1044
599 1045
590 1046
591 1047
592 1048
593 1049
594 1050
595 1051
596 1052
597 1053
598 1054
599 1055
590 1056
591 1057
592 1058
593 1059
594 1060
595 1061
596 1062
597 1063
598 1064
599 1065
590 1066
591 1067
592 1068
593 1069
594 1070
595 1071
596 1072
597 1073
598 1074
599 1075
590 1076
591 1077
592 1078
593 1079
594 1080
595 1081
596 1082
597 1083
598 1084
599 1085
590 1086
591 1087
592 1088
593 1089
594 1090
595 1091
596 1092
597 1093
598 1094
599 1095
590 1096
591 1097
592 1098
593 1099
594 1100
595 1101
596 1102
597 1103
598 1104
599 1105
590 1106
591 1107
592 1108
593 1109
594 1110
595 1111
596 1112
597 1113
598 1114
599 1115
590 1116
591 1117
592 1118
593 1119
594 1120
595 1121
596 1122
597 1123
598 1124
599 1125
590 1126
591 1127
592 1128
593 1129
594 1130
595 1131
596 1132
597 1133
598 1134
599 1135
590 1136
591 1137
592 1138
593 1139
594 1140
595 1141
596 1142
597 1143
598 1144
599 1145
590 1146
591 1147
592 1148
593 1149
594 1150
595 1151
596 1152
597 1153
598 1154
599 1155
590 1156
591 1157
592 1158
593 1159
594 1160
595 1161
596 1162
597 1163
598 1164
599 1165
590 1166
591 1167
592 1168
593 1169
594 1170
595 1171
596 1172
597 1173
598 1174
599 1175
590 1176
591 1177
592 1178
593 1179
594 1180
595 1181
596 1182
597 1183
598 1184
599 1185
590 1186
591 1187
592 1188
593 1189
594 1190
595 1191
596 1192
597 1193
598 1194
599 1195
590 1196
591 1197
592 1198
593 1199
594 1200
595 1201
596 1202
597 1203
598 1204
599 1205
590 1206
591 1207
592 1208
593 1209
594 1210
595 1211
596 1212
597 1213
598 1214
599 1215
590 1216
591 1217
592 1218
593 1219
594 1220
595 1221
596 1222
597 1223
598 1224
599 1225
590 1226
591 1227
592 1228
593 1229
594 1230
595 1231
596 1232
597 1233
598 1234
599 1235
590 1236
591 1237
592 1238
593 1239
594 1240
595 1241
596 1242
597 1243
598 1244
599 1245
590 1246
591 1247
592 1248
593 1249
594 1250
595 1251
596 1252
597 1253
598 1254
599 1255
590 1256
591 1257
592 1258
593 1259
594 1260
595 1261
596 1262
597 1263
598 1264
599 1265
590 1266
591 1267
592 1268
593 1269
594 1270
595 1271
596 1272
597 1273
598 1274
599 1275
590 1276
591 1277
592 1278
593 1279
594 1280
595 1281
596 1282
597 1283
598 1284
599 1285
590 1286
591 1287
592 1288
593 1289
594 1290
595 1291
596 1292
597 1293
598 1294
599 1295
590 1296
591 1297
592 1298
593 1299
594 1300
595 1301
596 1302
597 1303
598 1304
599 1305
590 1306
591 1307
592 1308
593 1309
594 1310
595 1311
596 1312
597 1313
598 1314
599 1315
590 1316
591 1317
592 1318
593 1319
594 1320
595 1321
596 1322
597 1323
598 1324
599 1325
590 1326
591 1327
592 1328
593 1329
594 1330
595 1331
596 1332
597 1333
598 1334
599 1335
590 1336
591 1337
592 1338
593 1339
594 1340
595 1341
596 1342
597 1343
598 1344
599 1345
590 1346
591 1347
592 1348
593 1349
594 1350
595 1351
596 1352
597 1353
598 1354
599 1355
590 1356
591 1357
592 1358
593 1359
594 1360
595 1361
596 1362
597 1363
598 1364
599 1365
590 1366
591 1367
592 1368
593 1369
594 1370
595 1371
596 1372
597 1373
598 1374
599 1375
590 1376
591 1377
592 1378
593 1379
594 1380
595 1381
596 1382
597 1383
598 1384
599 1385
590 1386
591 1387
592 1388
593 1389
594 1390
595 1391
596 1392
597 1393
598 1394
599 1395
590 1396
591 1397
592 1398
593 1399
594 1400
595 1401
596 1402
597 1403
598 1404
599 1405
590 1406
591 1407
592 1408
593 1409
594 1410
595 1411
596 1412
597 1413
598 1414
599 1415
590 1416
591 1417
592 1418
593 1419
594 1420
595 1421
596 1422
597 1423
598 1424
599 1425
590 1426
591 1427
592 1428
593 1429
594 1430
595 1431
596 1432
597 1433
598 1434
599 1435
590 1436
591 1437
592 1438
593 1439
594 1440
595 1441
596 1442
597 1443
598 1444
599 1445
590 1446
591 1447
592 1448
593 1449
594 1450
595 1451
596 1452
597 1453
598 1454
599 1455
590 1456
591 1457
592 1458
593 1459
594 1460
595 1461
596 1462
597 1463
598 1464
599 1465
590 1466
591 1467
592 1468
593 1469
594 1470
595 1471
596 1472
597 1473
598 1474
599 1475
590 1476
591 1477
592 1478
593 1479
594 1480
595 1481
596 1482
597 1483
598 1484
599 1485
590 1486
591 1487
592 1488
593 1489
594 1490
595 1491
596 1492
597 1493
598 1494
599 1495
590 1496
591 1497
592 1498
593 1499
594 1500
595 1501
596 1502
597 1503
598 1504
599 1505
590 1506
591 1507
592 1508
593 1509
594 1510
595 1511
596 1512
597 1513
598 1514
599 1515
590 1516
591 1517
592 1518
593 1519
594 1520
595 1521
596 1522
597 1523
598 1524
599 1525
590 1526
591 1527
592 1528
593 1529
594 1530
595 1531
596 1532
597 1533
598 1534
599 1535
590 1536
591 1537
592 1538
593 1539
594 1540
595 1541
596 1542
597 1543
598 1544
599 1545
590 1546
591 1547
592 1548
593 1549
594 1550
595 1551
596 1552
597 1553
598 1554
599 1555
590 1556
591 1557
592 1558
593 1559
594 1560
595 1561
596 1562
597 1563
598 1564
599 1565
590 1566
591 1567
592 1568
593 1569
594 1570
595 1571
596 1572
597 1573
598 1574
599 1575
590 1576
591 1577
592 1578
593 1579
594 1580
595 1581
596 1582
597 1583
598 1584
599 1585
590 1586
591 1587
592 1588
593 1589
594 1590
595 1591
596 1592
597 1593
598 1594
599 1595
590 1596
591 1597
592 1598
593 1599
594 1600
595 1601
596 1602
597 1603
598 1604
599 1605
590 1606
591 1607
592 1608
593 1609
594 1610
595 1611
596 1612
597 1613
598 1614
599 1615
590 1616
591 1617
592 1618
593 1619
594 1620
595 1621
596 1622
597 1623
598 1624
599 1625
590 1626
591 1627
592 1628
593 1629
594 1630
595 1631
596 1632
597 1633
598 1634
599 1635
590 1636
591 1637
592 1638
593 1639
594 1640
595 1641
596 1642
597 1643
598 1644
599 1645
590 1646
591 1647
592 1648
593 1649
594 1650
595 1651
596 1652
597 1653
598 1654
599 1655
590 1656
591 1657
592 1658
593 1659
594 1660
595 1661
596 1662
597 1663
598 1664
599 1665
590 1666
591 1667
592 1668
593 1669
594 1670
595 1671
596 1672
597 1673
598 1674
599 1675
590 1676
591 1677
592 1678
593 1679
594 1680
595 1681
596 1682
597 1683
598 1684
599 1685
590 1686
591 1687
592 1688
593 1689
594 1690
595 1691
596 1692
597 1693
598 1694
599 1695
590 1696
591 1697
592 1698
593 1699
594 1700
595 1701
596 1702
597 1703
598 1704
599 1705
590 1706
591 1707
592 1708
593 1709
594 1710
595 1711
596 1712
597 1713
598 1714
599 1715
590 1716
591 1717
592 1718
593 1719
594 1720
595 1721
596 1722
597 1723
598 1724
599 1725
590 1726
591 1727
592 1728
593 1729
594 1730
595 1731
596 1732
597 1733
598 1734
599 1735
590 1736
591 1737
592 1738
593 1739
594 1740
595 1741
596 1742
597 1743
598 1744
599 1745
590 1746
591 1747
592 1748
593 1749
594 1750
595 1751
596 1752
597 1753
598 1754
599 1755
590 1756
591 1757
592 1758
593 1759
594 1760
595 1761
596 1762
597 1763
598 1764
599 1765
590 1766
591 1767
592 1768
593 1769
594 1770
595 1771
596 1772
597 1773
598 1774
599 1775
590 1776
591 1777
592 1778
593 1779
594 1780
595 1781
596 1782
597 1783
598 1784
599 1785
590 1786
591 1787
592 1788
593 1789
594 1790
595 1791
596 1792
597 1793
598 1794
599 1795
590 1796
591 1797
592 1798
593 1799
594 1800
595 1801
596 1802
597 1803
598 1804
599 1805
590 1806
591 1807
592 1808
593 1809
594 1810
595 1811
596 1812
597 1813
598 1814
599 1815
590 1816

```

Constant-time code is hard to write

```
386 387     SSL3_NUERO();  
384 384     unsigned int mac_size;  
385 385     unsigned char mac[SHA_DIGEST_LENGTH];  
386 386     int decryption;  
387 387     EVP_DigestUpdate(&md_ctx, md, 2);  
388 388     EVP_DigestUpdate(&md_ctx, rec->input, rec->length);  
389 389     EVP_DigestFinal(&md_ctx, mac, &mac_len);  
390 390     if (decryption) {  
391 391         /* decrypt HMAC padding once */  
392 392         aesni_cbc_decrypt(in, out, len,  
393 393             &key->ks, ctx->iv, 0);  
394 394     }  
395 395     if (plen) { /* TLS mode of operation */  
396 396         /* figure out payload length */  
397 397         if (len<(size_t)(out[len-1]+1)*SHA_DIGEST_LENGTH))  
398 398             return 0;  
399 399     len -= (out[len-1]*SHA_DIGEST_LENGTH);  
400 400     size_t inp_len, mask, j, 1;  
401 401     unsigned int res, maxpad, pad, bitlen;  
402 402     int ret = 1;  
403 403     union { unsigned int u[SHA_LBLOCK];  
404 404         unsigned char c[SHA_CBLOCK]; }  
405 405     *data = (void *)key->md.data;  
406 406     if ((key->aux.tls_aad[plen-4]<>8|key->aux.tls_aad[plen-3])  
407 407         >= TLS1_1_VERSION) {  
408 408         len = AES_BLOCK_SIZE;  
409 409         >= TLS1_1_VERSION)  
410 410         iv = AES_BLOCK_SIZE;  
411 411     }  
412 412     key->aux.tls_aad[plen-2] = len>>8;  
413 413     key->aux.tls_aad[plen-1] = len;  
414 414     if (len<(iv*SHA_DIGEST_LENGTH+1))  
415 415         return 0;  
416 416     /* omit explicit iv */  
417 417     out += iv;  
418 418     len -= iv;  
419 419     /* figure out payload length */  
420 420     pad = out[len-1];  
421 421     maxpad = len-(SHA_DIGEST_LENGTH+1);  
422 422     maxpad |= (255-maxpad)>>((sizeof(maxpad)*8)-8);  
423 423     maxpad &= 255;  
424 424     /* calculate HMAC and verify it */  
425 425     key->aux.tls_aad[plen-2] = inp_len>>8;  
426 426     key->aux.tls_aad[plen-1] = inp_len;  
427 427     /* calculate HMAC */  
428 428     key->md = key->head;  
429 429     SHA1_Update(&key->md, key->aux.tls_aad, plen);  
430 430     SHA1_Update(&key->md, out+iv, len);  
431 431     SHA1_Final(mac, &key->d);  
432 432     len = SHA_DIGEST_LENGTH; /* amend mac */  
433 433     if (len>=(256*SHA_CBLOCK)) {  
434 434         j = (len-(256*SHA_CBLOCK))&(0-SHA_CBLOCK);  
435 435         j += SHA_CBLOCK-key->md.num;
```

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

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



```
y = -public & 42 | (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) {                                if (s) {  
    return 40;      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];
```



$O(n)$

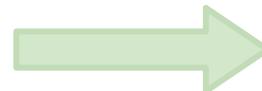
```
for (uint32_t i = 0; i < buffer.length(); i++) {  
    if (i == secret_index) {  
        x = buffer[i],  
    }  
}
```

A large red 'X' is drawn over the entire block of code, indicating it is inefficient.

Inefficient transformations

$O(1)$

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



$O(n)$

```
for (uint32_t i = 0; i < buffer.length(); i++) {  
    if (i == secret_index) {  
        x = buffer[i],  
    }  
}
```

A large red 'X' is drawn over the entire block of code, indicating it is inefficient.

Reject if transformation is inefficient

Unsafe transformations

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

Unsafe transformations

```
if (j < secret_len) {    →      x = -(j < secret_len) & arr[j]
    x = 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**

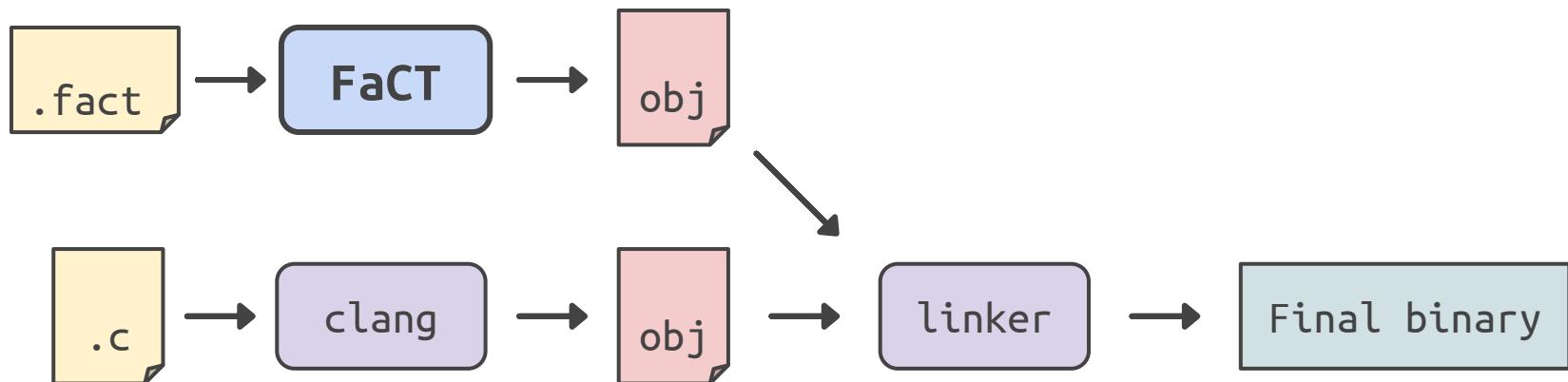
Porting code to FaCT

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



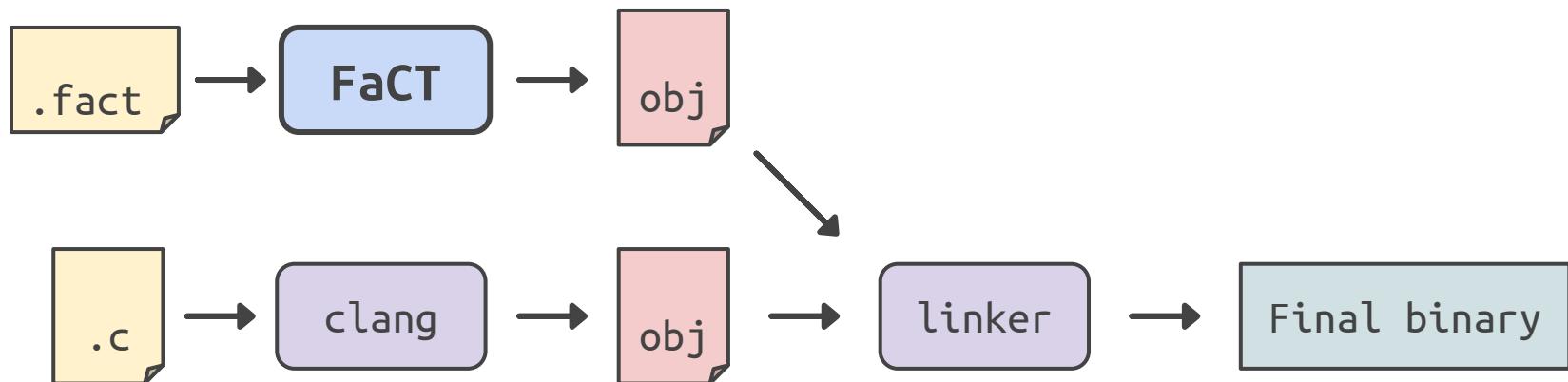
Porting code to FaCT

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



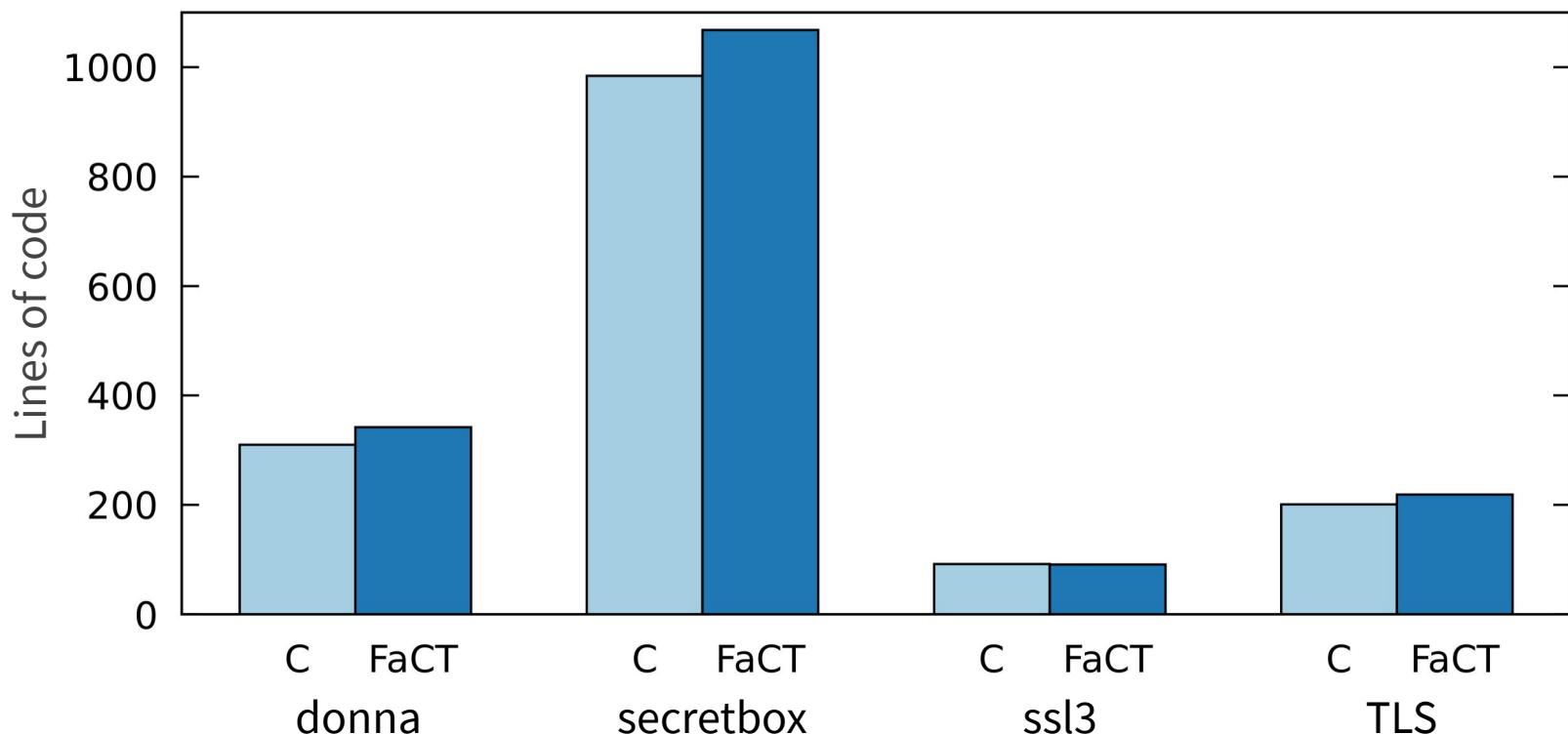
Porting code to FaCT

- 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

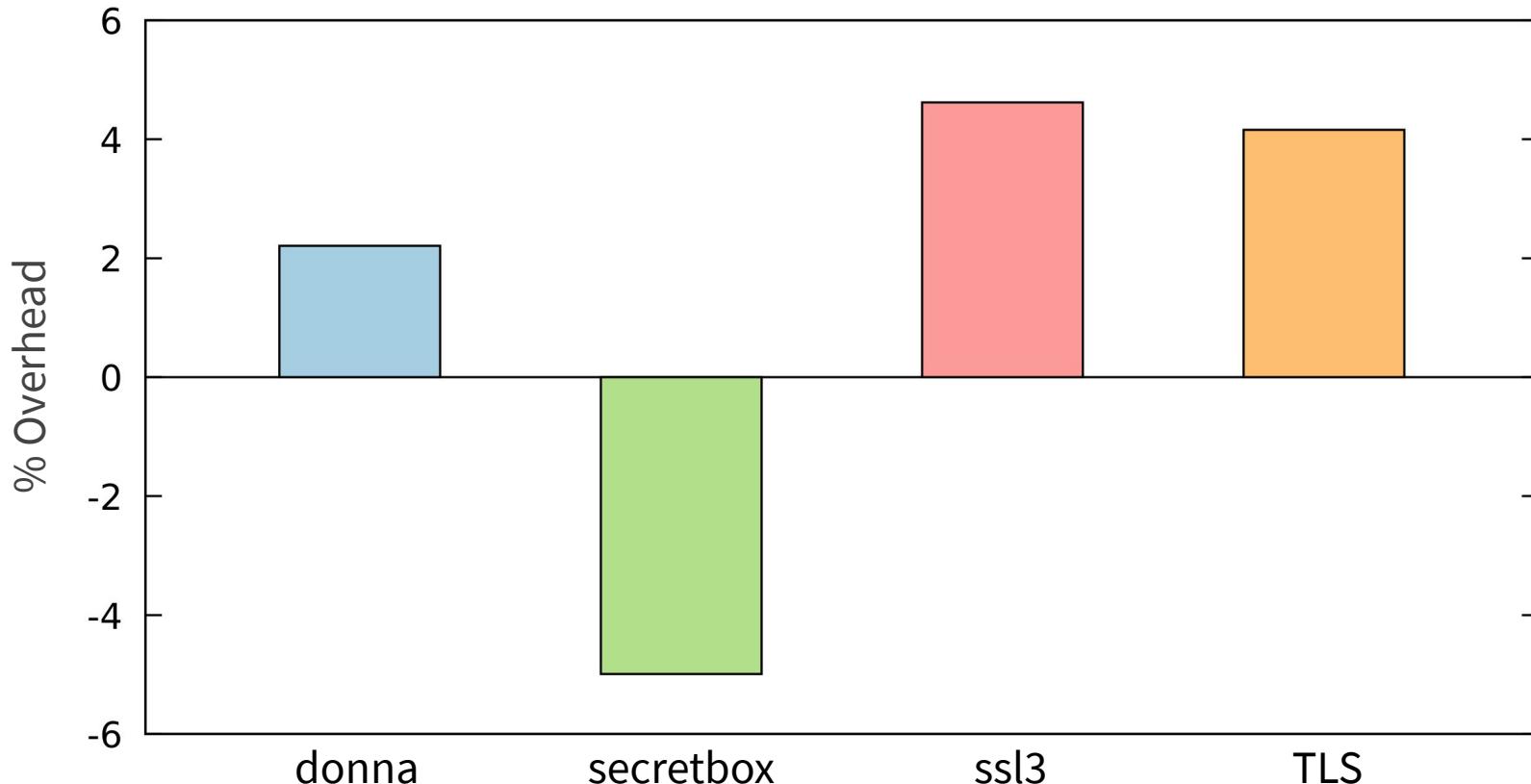
- Optimized with **same optimization flags**
- Empirically tested to be constant-time

Performance vs. C

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

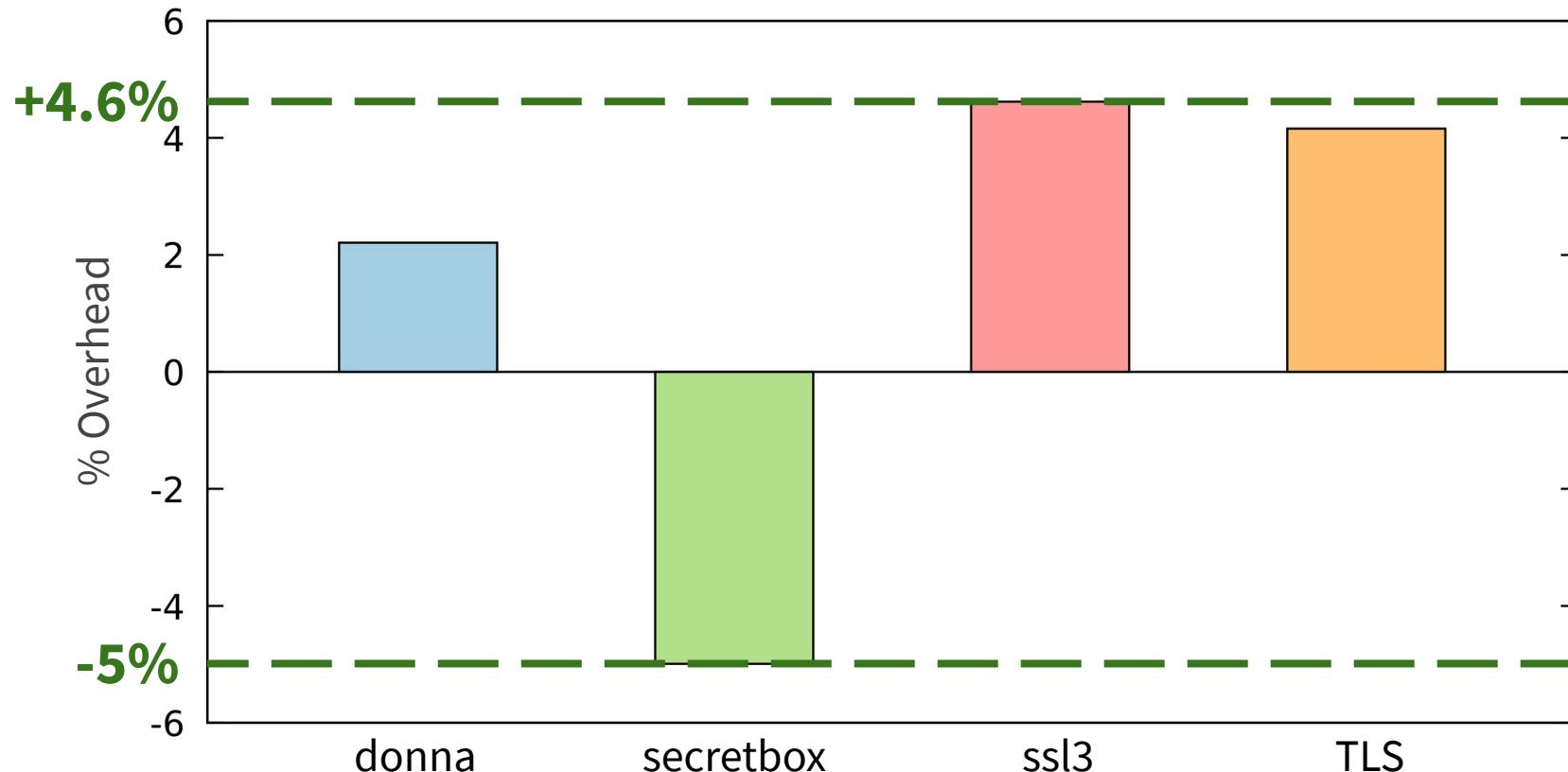
Performance vs. C

- 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?**

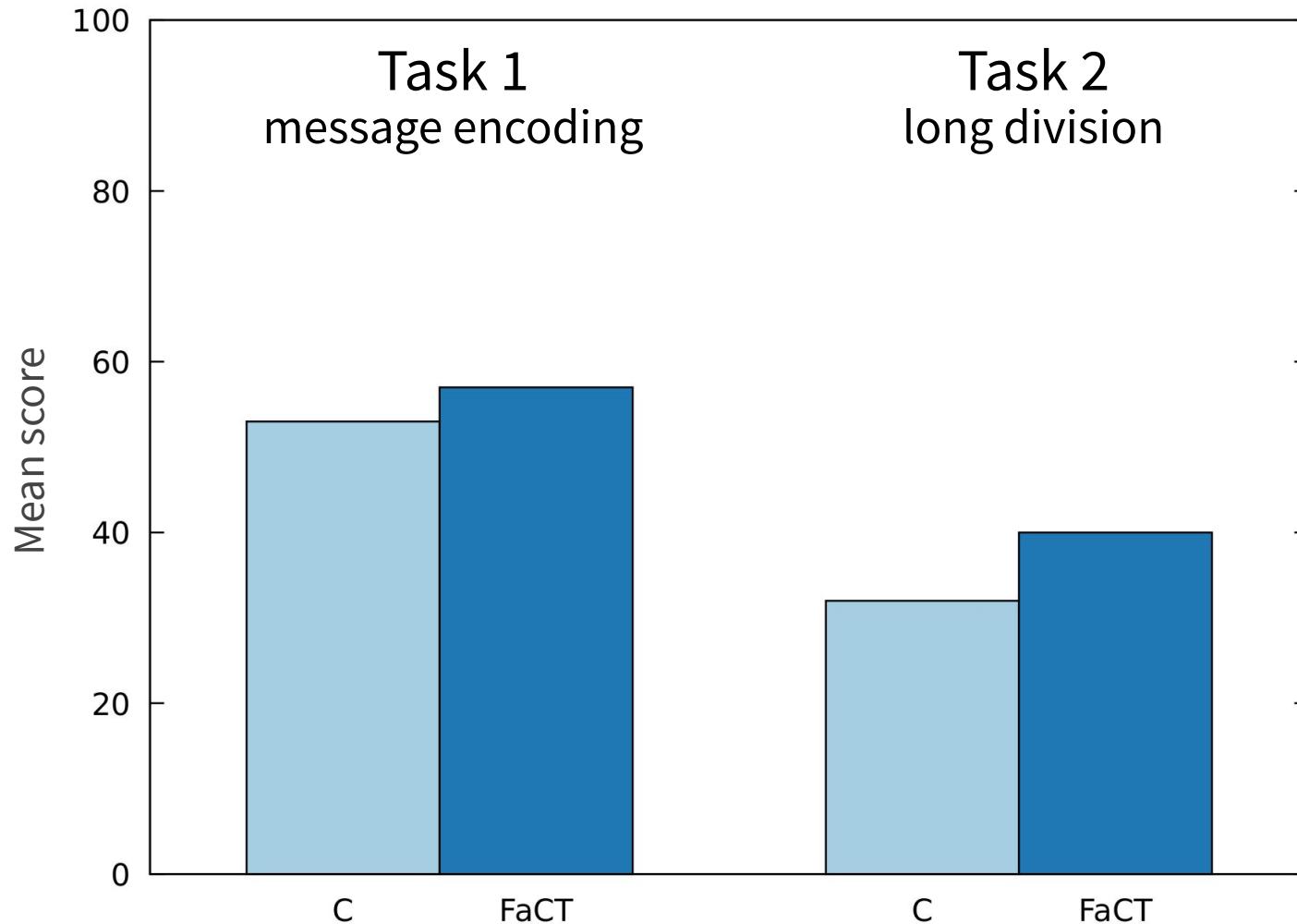
User study: FaCT vs. C

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

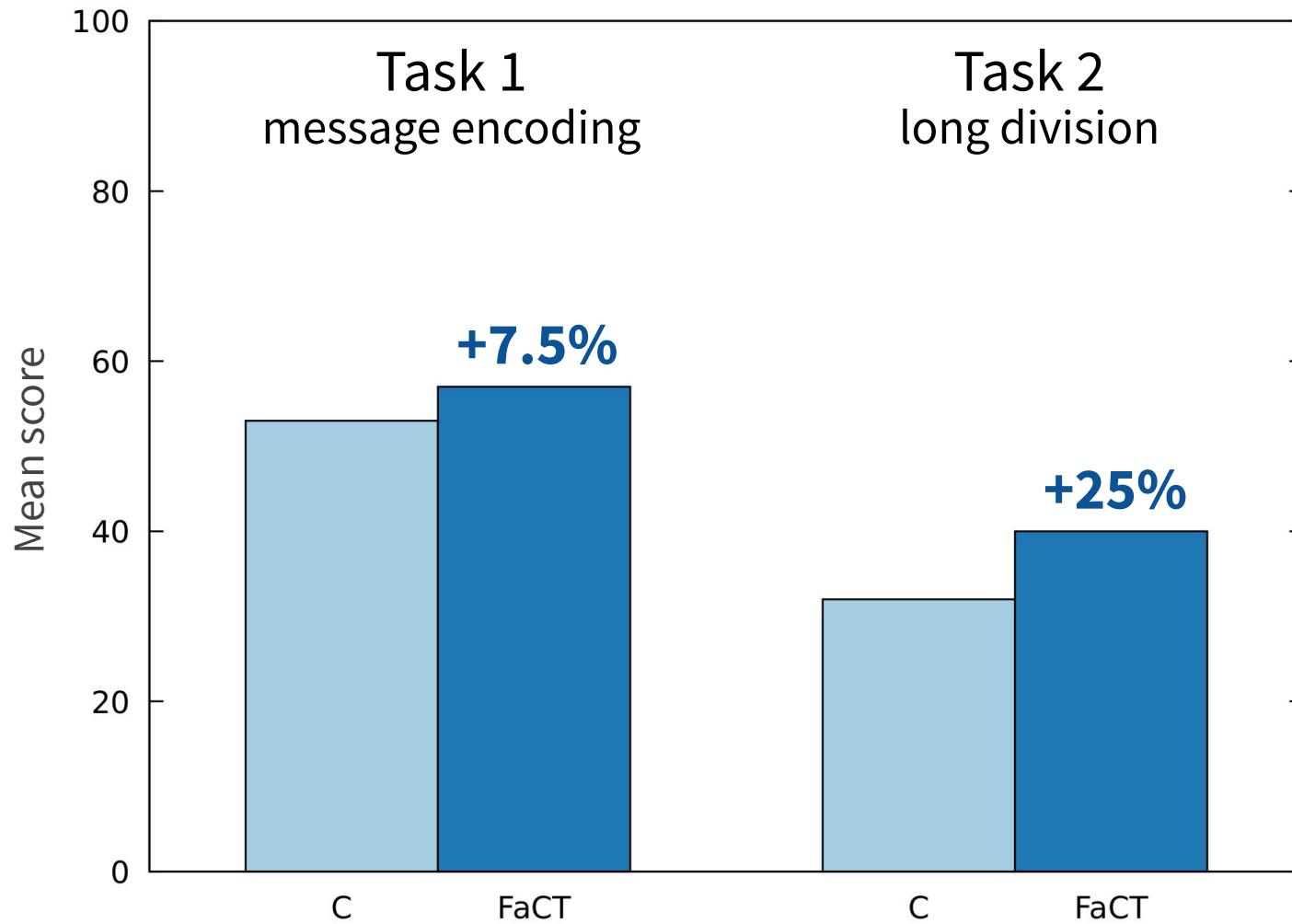
User study: FaCT vs. C

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

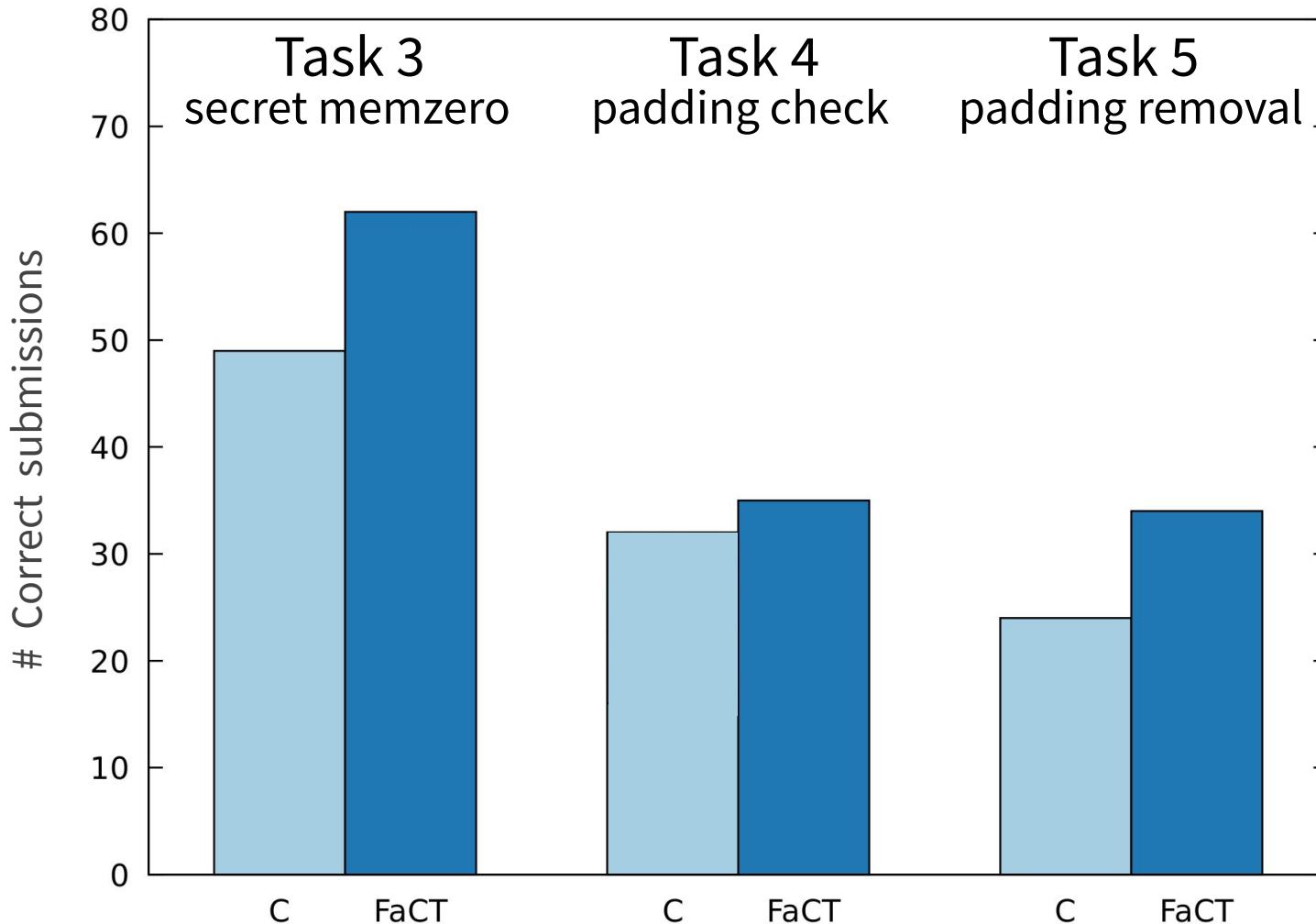
Understanding constant-time code



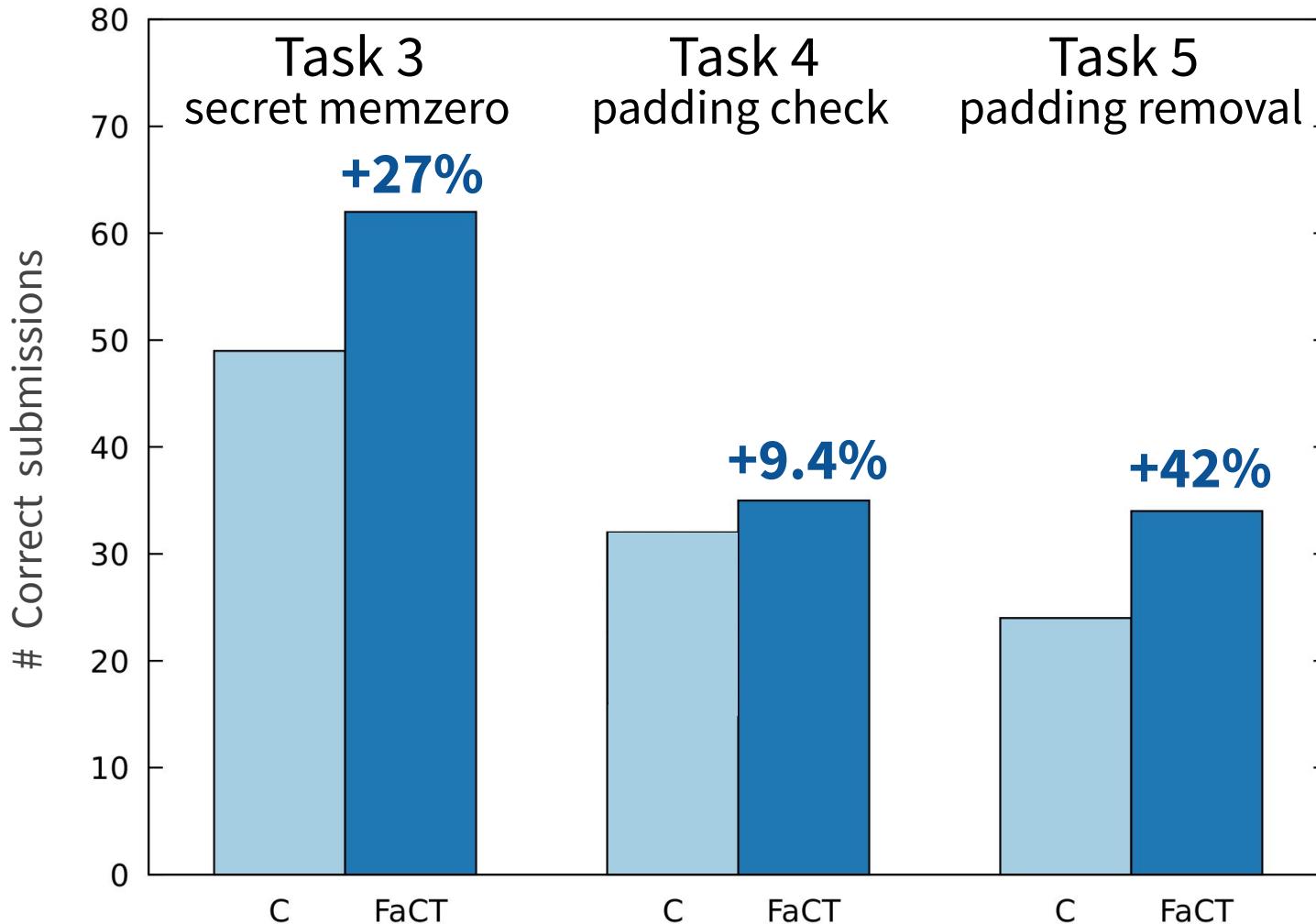
Understanding constant-time code



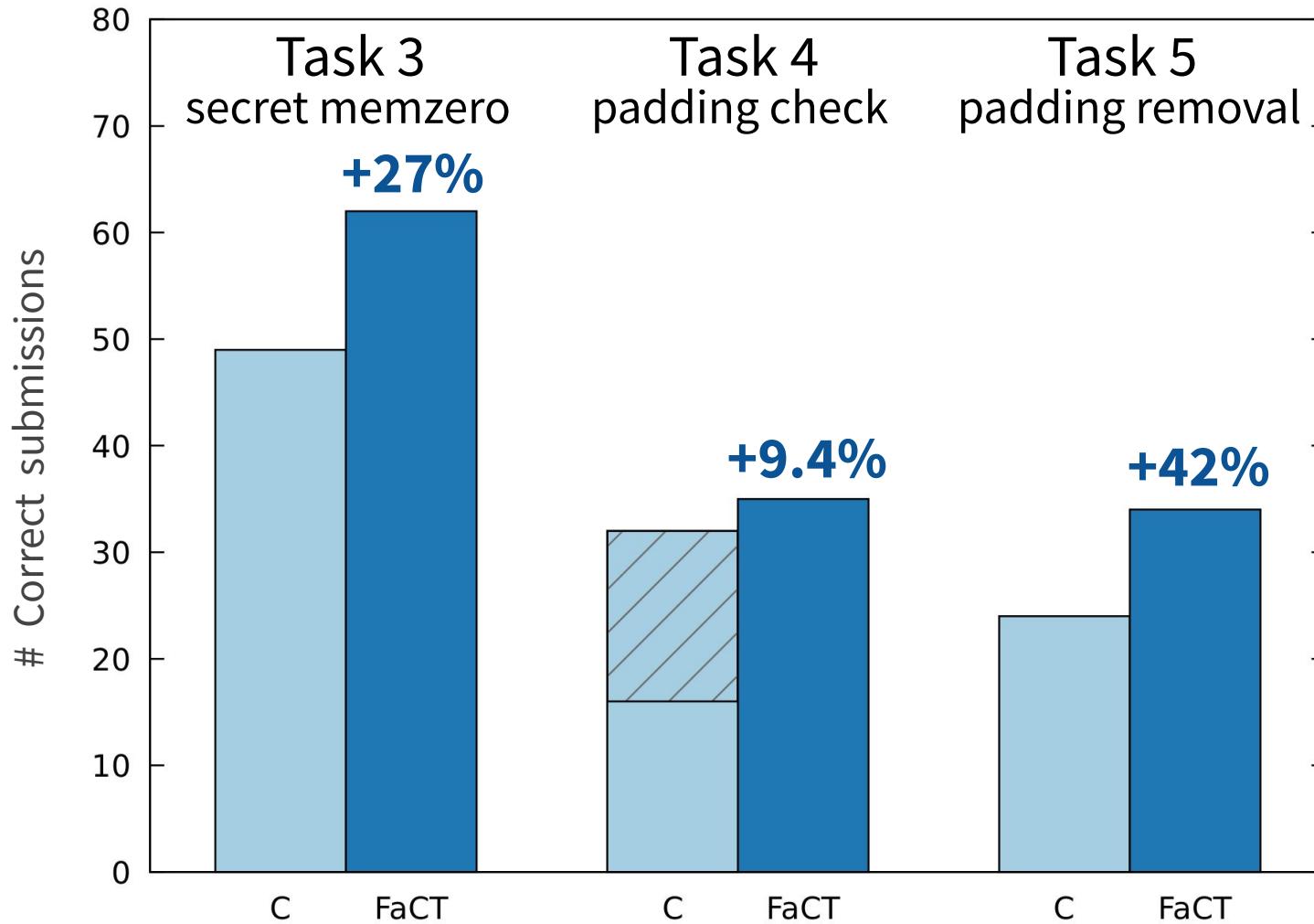
Writing constant-time code



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;
}
```