ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Σχολή Ψηφιακής Τεχνολογίας

Τμήμα Πληροφορικής και Τηλεματικής

Ομομορφική κρυπτογράφηση με τη χρήση της βιβλιοθήκης SEAL

Ιωάννης Σομός

Αθήνα, 5 Μαρτίου 2022

HAROKOPIO UNIVERSITY

School of Digital Technology

Department of Informatics and Telematics

Homomorphic encryption using the SEAL library

Ioannis Somos

Athens, 5th March 2022

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Σχολή Ψηφιακής Τεχνολογίας

Τμήμα Πληροφορικής και Τηλεματικής

Τριμελής Εξεταστική Επιτροπή

Παναγιώτης Ριζομυλιώτης (Επιβλέπων)

Επίκουρος Καθηγητής, Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο Αθηνών

Κωνσταντίνος Τσερπές

Αναπληρωτής Καθηγητής, Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο Αθηνών

Σωτήριος Ξύδης

Επίκουρος Καθηγητής, Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο Αθηνών

Ο Ιωάννης Σομός

δηλώνω υπεύθυνα ότι:

1. Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.
2. Αποδέχομαι ότι η ΒΚΠ μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από την ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

Περιεχόμενα

Περίληψη

Ο όρος «ομομορφική κρυπτογράφηση» αναφέρεται σε μία κατηγορία μεθόδων κρυπτογράφησης που οραματίστηκαν αρχικά οι Rivest, Adleman και Dertouzos το 1978. Η πρώτη υλοποίηση ενός συστήματος πλήρους ομομορφικής κρυπτογράφησης προτάθηκε από τον Gentry το 2009, τριάντα ένα χρόνια αργότερα. Η ομομορφική κρυπτογράφηση επιτρέπει την εκτέλεση πράξεων σε κρυπτογραφημένα δεδομένα δίχως να απαιτείται προηγουμένως η αποκρυπτογράφησή τους, εγγυώντας με αυτόν τον τρόπο την προστασία της εμπιστευτικότητάς τους. Το αποτέλεσμα των πράξεων επίσης παραμένει κρυπτογραφημένο και μπορεί να αποκαλυφθεί μόνο από τον κάτοχο του μυστικού κλειδιού που χρησιμοποιήθηκε κατά την κρυπτογράφηση. Η εργασία αυτή παρουσιάζει έναν αλγόριθμο πολλαπλασιασμού κρυπτογραφημένων τετραγωνικών πινάκων, υλοποιημένο στη γλώσσα C++ με χρήση της βιβλιοθήκης 'SEAL' και το νέας γενιάς σύστημα ομομορφικής κρυπτογράφησης 'CKKS'.

Λέξεις Κλειδιά: *κρυπτογραφία, ομομορφική κρυπτογράφηση, πολλαπλασιασμός πινάκων*

Abstract

The term 'homomorphic encryption' refers to a class of encryption methods originally envisioned by Rivest, Adleman & Dertouzos in 1978. The first implementation of a fully homomorphic encryption scheme was proposed by Gentry in 2009, thirty-one years later. Homomorphic encryption allows for operations on encrypted data without requiring them to be decrypted beforehand, thus guaranteeing their confidentiality. The result of the computations also remains encrypted and can only be revealed by the owner of the secret key used during the encryption. This thesis presents a multiplication algorithm for encrypted square matrices which was implemented in the C++ programming language using the 'SEAL' library and the new generation homomorphic encryption scheme 'CKKS'.

Keywords: *cryptography, homomorphic encryption, matrix multiplication*

Κατάλογος σχημάτων

Κατάλογος πινάκων

Εισαγωγή

Η ομομορφική κρυπτογράφηση είναι μία ειδική μορφή κρυπτογραφίας δημοσίου κλειδιού η οποία επιτρέπει την απευθείας εκτέλεση συγκεκριμένων υπολογισμών πάνω σε κρυπτογραφημένα δεδομένα. Βασίζεται στη μαθηματική έννοια του ομομορφισμού (σχήμα 1) που υποδεικνύει ότι ορισμένες πράξεις σε κρυπτοκείμενα αναλογούν σε πράξεις στα αντίστοιχα απλά κείμενα. [Ben20]

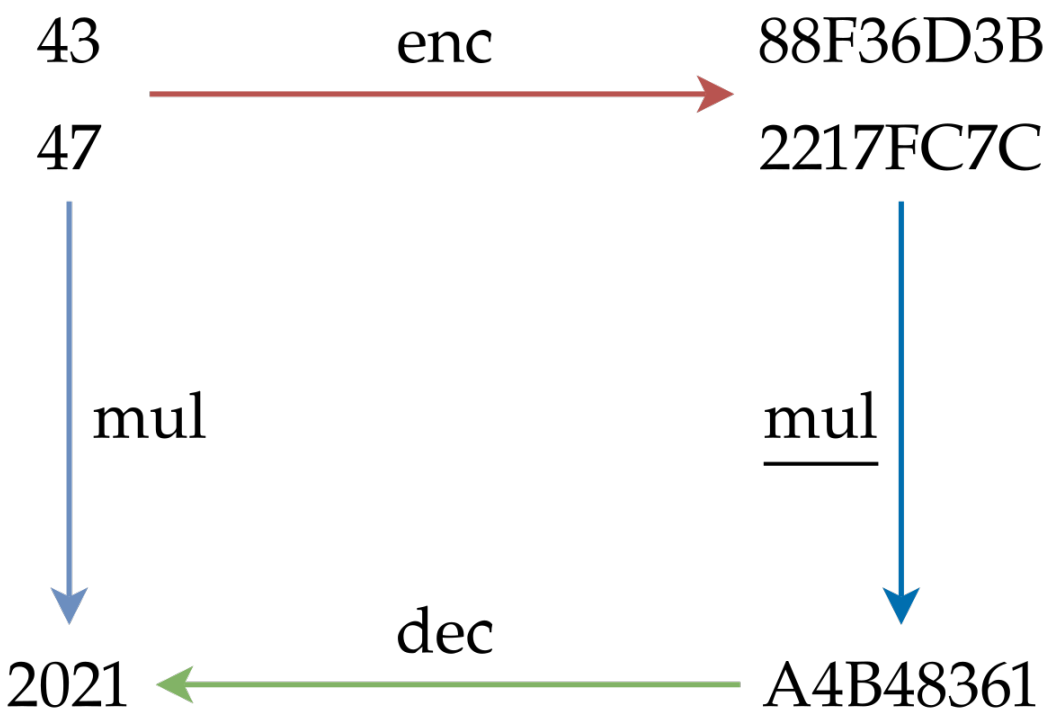
Διακρίνεται στις εξής κατηγορίες με βάση τον τύπο και το πλήθος των πράξεων που έχει τη δυνατότητα να εκτελέσει το σύστημα: [Aca+18]

Μερικώς Ομομορφική Κρυπτογράφηση
Επιτρέπει την εκτέλεση απεριόριστου αριθμού εκτελέσεων μίας συγκεκριμένης πράξης.

Κάπως Ομομορφική Κρυπτογράφηση
Επιτρέπει την εκτέλεση πολλαπλών πράξεων, αλλά για περιορισμένο αριθμό εκτελέσεων.

Πλήρως Ομομορφική Κρυπτογράφηση
Επιτρέπει την εκτέλεση οποιασδήποτε πράξης και για αυθαίρετο αριθμό εκτελέσεων.

Σχήμα 1: Ομομορφικός πολλαπλασιασμός



Η βιβλιοθήκη SEAL παρέχει τη δυνατότητα εκτέλεσης συγκεκριμένων πράξεων σε κρυπτογραφημένα διανύσματα. Η εργασία αυτή επεκτείνει τη βιβλιοθήκη, υλοποιώντας ένα πρόγραμμα για τον πολλαπλασιασμό κρυπτογραφημένων πινάκων.

Δομή

Η παρούσα πτυχιακή εργασία χωρίζεται σε έξι κεφάλαια.

- Στο παρόν εισαγωγικό κεφάλαιο δίνονται ο ορισμός και οι κατηγορίες της ομομορφικής κρυπτογράφησης, καθώς και μία περιγραφή του πλαισίου της εργασίας.
- Το δεύτερο κεφάλαιο ξεκινάει με μία αναδρομή στην ιστορία της ομομορφικής κρυπτογράφησης, συνεχίζει αναλύοντας το πρότυπο λειτουργίας της, και στο τέλος περιγράφει συγκεκριμένα το σύστημα CKKS και τη βιβλιοθήκη SEAL που χρησιμοποιήθηκαν παρακάτω.
- Το τρίτο κεφάλαιο αναλύει τον ομομορφικό αλγόριθμο πολλαπλασιασμού κρυπτογραφημένων πινάκων που πρότειναν οι Jiang, Kim, Lauter και Song, παραθέτοντας ένα παράδειγμα λειτουργίας του.
- Το τέταρτο κεφάλαιο περιγράφει την υλοποίηση ενός προγράμματος στη γλώσσα C++ για τον πολλαπλασιασμό κρυπτογραφημένων πινάκων με τη χρήση του παραπάνω αλγορίθμου.
- Το πέμπτο κεφάλαιο περιέχει τα αποτελέσματα της εργασίας και πιθανούς τρόπους βελτίωσης της υλοποίησης.
- Στο τελευταίο κεφάλαιο αναφέρονται μερικές εφαρμογές της ομομορφικής κρυπτογράφησης καθώς και τα συμπεράσματα που προέκυψαν.

Ομομορφική Κρυπτογράφηση

Η ομομορφική κρυπτογράφηση έχει κινήσει τα τελευταία χρόνια το ενδιαφέρον της επιστήμης των δεδομένων, αλλά ερευνάται από κρυπτογράφους εδώ και πολλές δεκαετίες.

Την ιδέα οραματίστηκαν αρχικά οι Rivest, Adleman και Dertouzos το 1978 ως «ομομορφισμοί ιδιωτικότητας» [RAD78]. Ακολούθησε μία περίοδος τριάντα ετών χωρίς την επίτευξη κάποιου συστήματος FHE, κατά την οποία όμως αναπτύχθηκαν διάφορα συστήματα PHE και SHE. Ορισμένα αξιοσημείωτα συστήματα PHE είναι τα κρυπτοσυστήματα RSA [RSA78] και ElGamal [ElG85] που εκτελούν αυθαίρετο αριθμό πολλαπλασιασμών, καθώς και το κρυπτοσύστημα Paillier [Pai99] το οποίο εκτελεί αυθαίρετο αριθμό προσθέσεων. Παρομοίως, μερικά σημαντικά συστήματα SHE είναι τα αλλοιωμένα (garbled) κυκλώματα[1] του Yao [Yao82, ZPG15 §9.4.5] τα οποία λειτουργούν μόνο για δύο τιμές εισόδου δίχως μάλιστα δυνατότητα επαναχρησιμοποίησης των κυκλωμάτων, και το κρυπτοσύστημα BGN [BGN05] που μπορεί να εκτελέσει αυθαίρετο αριθμό προσθέσεων και, επιπλέον, έναν πολλαπλασιασμό.

Το πρώτο πλήρως ομομορφικό κρυπτοσύστημα προτάθηκε από τον Gentry το 2009 βασισμένο σε «ιδεώδη» δικτυωτά και κατασκευασμένο με βάση ένα κάπως ομομορφικό κρυπτοσύστημα. [Gen09] Η διαδικασία της κρυπτογράφησης προσθέτει θόρυβο στο αρχικό μήνυμα ο οποίος μπορεί να αφαιρεθεί με την αποκρυπτογράφηση του παραγόμενου κρυπτοκειμένου. Όμως, η κάθε αποτίμηση κάποιας συνάρτησης επάνω στα κρυπτοκείμενα αυξάνει τον θόρυβο θέτοντας ένα όριο στο βάθος του υπολογιστικού κυκλώματος. Ο Gentry έλυσε αυτό το πρόβλημα τροποποιώντας το σύστημα ώστε να γίνει «εκκινήσιμο» (bootstrappable), δηλαδή να μπορεί να υπολογίσει τη δική του συνάρτηση αποκρυπτογράφησης και τουλάχιστον μία άλλη συνάρτηση. Έδειξε, έπειτα, πως οποιοδήποτε εκκινήσιμο σύστημα PHE μπορεί να μετατραπεί σε FHE ενσωματώνοντας αναδρομικά τον εαυτό του ώστε να ανανεώνει το κρυπτοκείμενο όποτε ο θόρυβος αυξάνεται πολύ, επιτυγχάνοντας έτσι τον υπολογισμό αυθαίρετου αριθμού συναρτήσεων χωρίς μεγάλη αύξηση του θορύβου. [ZPG15 §12.5.2] Το σύστημα αυτό απλουστεύθηκε τον επόμενο χρόνο αντικαθιστώντας τα ιδεώδη δικτυωτά με στοιχειώδη αριθμητική υπολοίπων. [Dij+10]

Αργότερα, εμφανίστηκαν πολλά νέα ομομορφικά συστήματα βασισμένα σε δύο δύσκολα υπολογιστικά προβλήματα: το LWE [Reg05], ή την ειδίκευσή του (RLWE [LPR13]), και μία παραλλαγή του NTRU [HPS98]. Στο πρώτο πρόβλημα βασίστηκαν τα συστήματα BGV [BGV12] (2011), BFV [BFV12] (2012), GSW [GSW13] (2013) και CKKS [CKKS17] (2017)—το οποίο θα δούμε πιο αναλυτικά παρακάτω. Τα συστήματα BGV και BFV εμφανίζουν πολύ χαμηλότερο ρυθμό αύξησης θορύβου και μπορούν να λειτουργήσουν έως κάποιο υπολογιστικό επίπεδο χωρίς τη διαδικασία εκκίνησης. Το σύστημα GSW, εκτός από τη μείωση του θορύβου, κατάφερε επίσης να αποφύγει ένα υπολογιστικά ακριβό βήμα επαναγραμμικοποίησης χρησιμοποιώντας μία νέα τεχνική «προσεγγιστικού ιδιοδιανύσματος». Στο δεύτερο πρόβλημα βασίστηκαν τα συστήματα LTV [LTV12] (2012) και BLLN [BLLN13] (2013) τα οποία προσφέρουν παρόμοια πλεονεκτήματα με τα BGV και BFV αλλά η παραλλαγή του NTRU που αξιοποίησαν αποδείχθηκε μη ασφαλής [ABD16] οπότε δε χρησιμοποιούνται.

[1] Αποτελούνται από λογικές πύλες με κωδικοποιημένους πίνακες αληθείας.

Πρότυπο λειτουργίας

Υπάρχει μία (πρώιμη) προσπάθεια από πολλαπλούς τεχνολογικούς φορείς να προτυποποιηθεί η ομομορφική κρυπτογράφηση ώστε να διευκολυνθεί η χρήση της από διάφορους τομείς. [Alb+18] Αυτό το πρότυπο περιγράφεται στο παρόν κεφάλαιο.

Αλγόριθμοι

Το πρότυπο καθορίζει αρχικά τους παρακάτω αλγορίθμους: [Alb+18 §1.1.1]

$\texttt{ParamGen}(\lambda, \mathcal{M}, k, b) \to \mathrm{P}$
: Αρχικοποιεί τις παραμέτρους (P) που χρησιμοποιούνται σε κάποιους από τους επόμενους αλγορίθμους—όπως συντελεστές υπολοίπων ή δακτυλίους—λαμβάνοντας ως είσοδο τα ακόλουθα ορίσματα:

  λ Συμβολίζει το επιθυμητό επίπεδο ασφαλείας (bit) του συστήματος.

  $\mathcal{M}$ Αναπαριστά τον χώρο των plaintext μηνυμάτων. Το πρότυπο ορίζει δύο[2] τύπους αυτού του χώρου: των ακεραίων υπολοίπων και των δακτυλίων.

  - Ο πρώτος τύπος παραμετροποιείται με έναν συντελεστή υπολοίπου, δηλαδή ισχύει $\mathcal{M} = \mathbb{Z}_p$. Συνεπώς, κάθε στοιχείο του χώρου είναι ένας ακέραιος που ανήκει στο διάστημα $[0, p)$ και όλες οι πράξεις εκτελούνται $\bmod\ p$.
  - Ο δεύτερος τύπος παραμετροποιείται επίσης με έναν συντελεστή υπολοίπου αλλά και ένα πολυώνυμο $f(x) \in \mathbb{Z}_p$, ώστε $\mathcal{M} = \mathbb{Z}_p[x]/f(x)$. Αυτό σημαίνει πως κάθε στοιχείο του χώρου είναι ένα πολυώνυμο με βαθμό μικρότερο του $f(x)$ και ακέραιους συντελεστές που ανήκουν στο διάστημα $[1, p-1)$. Οι πράξεις, εδώ, εκτελούνται $\bmod\ f(x)$ και $\bmod\ p$.

  k Υποδηλώνει το μέγεθος των διανυσμάτων προς κρυπτογράφηση.

  b Είναι μία βοηθητική παράμετρος που περιορίζει την πολυπλοκότητα των ομομορφικών υπολογισμών. Όσο μικρότερη είναι αυτή η παράμετρος, τόσο μικρότερες είναι και οι παράμετροι που παράγει ο αλγόριθμος· κάτι που οδηγεί σε πιο αποδοτικούς υπολογισμούς καθώς και τα κρυπτοκείμενα γίνονται μικρότερα. Φυσικά, πιο πολύπλοκα προγράμματα απαιτούν μεγαλύτερη τιμή.

$\texttt{PubKeygen}(\mathrm{P}) \to \mathrm{K}_s, \mathrm{K}_p, \mathrm{K}_e$
: Δημιουργεί ένα ζεύγος κλειδιών, μυστικού (K_s) και δημοσίου (K_p), καθώς και ένα επιπλέον κλειδί αποτίμησης (K_e). Το μυστικό κλειδί θα πρέπει να διατηρείται κρυφό από έναν χρήστη και μπορεί να χρησιμοποιηθεί για την αποκρυπτογράφηση κρυπτοκειμένων, ενώ το δημόσιο κλειδί μπορεί να κοινοποιηθεί και να χρησιμοποιηθεί από οποιονδήποτε για την κρυπτογράφηση μηνυμάτων. Το κλειδί αποτίμησης πρέπει να δοθεί σε οποιαδήποτε οντότητα αναμένεται να εκτελέσει ομομορφικές πράξεις πάνω στα κρυπτοκείμενα. Το δημόσιο κλειδί και το κλειδί αποτίμησης δεν επαρκούν για να λάβει κάποιος πληροφορίες από τα κρυπτοκείμενα χωρίς το μυστικό κλειδί.

$\texttt{SecKeygen}(\mathrm{P}) \to \mathrm{K}_s, \mathrm{K}_e$
: Αξιοποιείται από τον παραπάνω αλγόριθμο για τη δημιουργία του μυστικού κλειδιού (K_s) και του κλειδιού αποτίμησης (K_e).

$\texttt{PubEncrypt}(\mathrm{K}_p, \mathrm{M}) \to \mathrm{C}$
: Λαμβάνει ως είσοδο το δημόσιο κλειδί του συστήματος (K_p) και οποιοδήποτε $\mathrm{M} \in \mathcal{M}$ και παράγει από αυτό το κρυπτοκείμενο C.

[2] Προβλέπεται να ενταχθεί στο μέλλον και ένας τρίτος τύπος, των προσεγγιστικών αριθμών.

$\texttt{SecEncrypt}(\mathrm{K}_s, \mathrm{M}) \rightarrow \mathrm{C}$

Λαμβάνει ως είσοδο το μυστικό κλειδί του συστήματος (K_s) και οποιοδήποτε $\mathrm{M} \in \mathcal{M}$ και παράγει από αυτό το κρυπτοκείμενο C.

$\texttt{Decrypt}(\mathrm{K}_s, \mathrm{C}) \rightarrow \mathrm{M}$

Λαμβάνει ως είσοδο το μυστικό κλειδί του συστήματος (K_s) και κάποιο κρυπτοκείμενο C και το αποκωδικοποιεί στο αρχικό μήνυμα $\mathrm{M} \in \mathcal{M}$. Ενδέχεται η έξοδός του να είναι ένα ειδικό σύμβολο FAIL στην περίπτωση που αποτύχει η αποκρυπτογράφηση.

$\texttt{EvalAdd}(\mathrm{P}, \mathrm{K}_e, \mathrm{C}_1, \mathrm{C}_2) \rightarrow \mathrm{C}_3$

Προσθέτει τα κρυπτοκείμενα C_1 και C_2, λαμβάνοντας επίσης ως είσοδο τις παραμέτρους του συστήματος (P) και το κλειδί αποτίμησης (K_e), και δίνει ως αποτέλεσμα το κρυπτοκείμενο C_3. Η ορθότητά του επαληθεύεται εάν το C_3 είναι κρυπτογράφημα του $\mathrm{M}_1 + \mathrm{M}_2$, όπου M_1 και M_2 είναι τα αρχικά μηνύματα των C_1 και C_2 αντίστοιχα.

$\texttt{EvalAddConst}(\mathrm{P}, \mathrm{K}_e, \mathrm{C}_1, \mathrm{M}_2) \rightarrow \mathrm{C}_3$

Προσθέτει το κρυπτοκείμενο C_1 και το μήνυμα M_2, λαμβάνοντας επίσης ως είσοδο τις παραμέτρους του συστήματος (P) και το κλειδί αποτίμησης (K_e), και δίνει ως αποτέλεσμα το κρυπτοκείμενο C_3. Η ορθότητά του επαληθεύεται εάν το C_3 είναι κρυπτογράφημα του $\mathrm{M}_1 + \mathrm{M}_2$, όπου M_1 είναι το αρχικό μήνυμα του C_1.

$\texttt{EvalMult}(\mathrm{P}, \mathrm{K}_e, \mathrm{C}_1, \mathrm{C}_2) \rightarrow \mathrm{C}_3$

Πολλαπλασιάζει τα κρυπτοκείμενα C_1 και C_2, λαμβάνοντας επίσης ως είσοδο τις παραμέτρους του συστήματος (P) και το κλειδί αποτίμησης (K_e), και δίνει ως αποτέλεσμα το κρυπτοκείμενο C_3. Η ορθότητά του επαληθεύεται εάν το C_3 είναι κρυπτογράφημα του $\mathrm{M}_1 \cdot \mathrm{M}_2$, όπου M_1 και M_2 είναι τα αρχικά μηνύματα των C_1 και C_2 αντίστοιχα.

$\texttt{EvalMultConst}(\mathrm{P}, \mathrm{K}_e, \mathrm{C}_1, \mathrm{M}_2) \rightarrow \mathrm{C}_3$

Πολλαπλασιάζει το κρυπτοκείμενο C_1 και το μήνυμα M_2, λαμβάνοντας επίσης ως είσοδο τις παραμέτρους του συστήματος (P) και το κλειδί αποτίμησης (K_e), και δίνει ως αποτέλεσμα το κρυπτοκείμενο C_3. Η ορθότητά του επαληθεύεται εάν το C_3 είναι κρυπτογράφημα του $\mathrm{M}_1 \cdot \mathrm{M}_2$, όπου M_1 είναι το αρχικό μήνυμα του C_1.

$\texttt{Refresh}(\mathrm{P}, \mathit{mode}, \mathrm{K}_e, \mathrm{C}_1) \rightarrow \mathrm{C}_2$

Λαμβάνει ως είσοδο τις παραμέτρους του συστήματος (P), τη λειτουργία (mode) που πρόκειται να εκτελέσει, το κλειδί αποτίμησης (K_e) και ένα κρυπτοκείμενο C_1. Παράγει το κρυπτοκείμενο C_2 από το C_1 ανάλογα με τη λειτουργία η οποία μπορεί να είναι μία από τις εξής: 'Relinearise' (επαναγραμμικοποίηση), 'ModSwitch' (αλλαγή συντελεστή υπολοίπου), 'Bootstrap' (εκκίνηση). Η ορθότητα του αλγορίθμου επαληθεύεται εάν το C_2 είναι κρυπτογράφημα του ίδιου μηνύματος M_1 όπως το C_1. Ουσιαστικά, μετατρέπει ένα σύνθετο κρυπτοκείμενο σε ένα απλούστερο που κρυπτογραφεί το ίδιο αρχικό μήνυμα, είτε μειώνοντας τον θόρυβο σε κάποιο σταθερό επίπεδο ή δημιουργώντας ένα νέο κρυπτοκείμενο με άλλο μυστικό κλειδί.

$\texttt{ValidityCheck}(\mathrm{P}, \mathrm{K}_e, [\mathrm{C}_1, \ldots, \mathrm{C}_n], \mathfrak{F}) \rightarrow \mathit{flag}$

Λαμβάνει ως είσοδο τις παραμέτρους του συστήματος (P), το κλειδί αποτίμησης (K_e), μία λίστα κρυπτοκειμένων ($[\mathrm{C}_1, \ldots, \mathrm{C}_n]$), και μία προδιαγραφή ($\mathfrak{F}$) ενός προγράμματος ομομορφικών υπολογισμών. Ο αλγόριθμος αυτός είναι ντετερμινιστικός και δίνει ένα Boolean αποτέλεσμα ($\mathit{flag} \in \{0, 1\}$). Η ορθότητά του επαληθεύεται εάν για $\mathit{flag} = 1$ το αποτέλεσμα της εκτέλεσης του προγράμματος στη δοθείσα λίστα κρυπτοκειμένων αποκρυπτογραφείται σωστά.

Ιδιότητες

Για την ικανοποίηση των ιδιοτήτων ασφαλείας [Alb+18 §1.1.2] του συστήματος, απαιτείται η τυχαιότητα των αλγορίθμων $*\texttt{Encrypt}$ και $\texttt{Eval}*$ καθώς και του αλγορίθμου $\texttt{Refresh}$. Τα κλειδιά K επιλέγονται επίσης τυχαία (στην πράξη συνήθως με βάση την κατανομή ενός δακτυλίου από τις παραμέτρους P [Alb+18 §1.1.3]).

Σημασιολογική Ασφάλεια (IND-CPA)
: Ένα σύστημα ομομορφικής κρυπτογράφησης θεωρείται ασφαλές εάν δε δίνει τη δυνατότητα σε κάποιον να μαντέψει με πιθανότητα μεγαλύτερη του 50% αν κάποιο κρυπτοκείμενο προέρχεται από ένα εκ δύο εξίσου πιθανών διακριτών μηνυμάτων. Το κρυπτοκείμενο, λοιπόν, δεν πρέπει να διατηρεί καμία πληροφορία που αφορά το αρχικό μήνυμα.

Συμπάγεια
: Το σύστημα εγγυάται πως οι ομομορφικές πράξεις στα κρυπτοκείμενα δε διαστέλλουν το μέγεθός τους. Δηλαδή, ένας αποτιμητής μπορεί να εκτελέσει μία αυθαίρετη λίστα συναρτήσεων αποτίμησης και να λάβει ένα κρυπτοκείμενο ανεξάρτητα από την πολυπλοκότητα των συναρτήσεων αυτών.

Αποδοτική αποκρυπτογράφηση
: Το σύστημα εγγυάται επίσης ότι ο χρόνος εκτέλεσης της αποκρυπτογράφησης δεν εξαρτάται από τις συναρτήσεις που αποτιμήθηκαν στα κρυπτοκείμενα.

Επιπλέον, το πρότυπο αναφέρει μερικές ακόμη ιδιότητες [Alb+18 §1.1.6] οι οποίες δεν είναι μεν αναγκαίες για τη λειτουργία του συστήματος, αλλά μπορεί να είναι επιθυμητές υπό ορισμένες συνθήκες.

- Δυνατότητα κατανεμημένης εκτέλεσης υπολογισμών (από πολλαπλούς χρήστες).
- Αποτροπή ενεργών επιθέσεων στο σύστημα. (π.χ. αντικατάσταση κρυπτοκειμένου)
- Απόκρυψη των υπολογισμών που εκτελέστηκαν σε ένα κρυπτοκείμενο.
- Δυνατότητα «εξέλιξης» (αλλαγής) του μυστικού κλειδιού σε περίπτωση διακινδύνευσης.
- Ανθεκτικότητα απέναντι σε επιθέσεις πλαγίου μονοπατιού. (π.χ. επίθεση χρόνου)
- Υλοποίηση κρυπτογραφίας βάσει ταυτότητας (IBE).

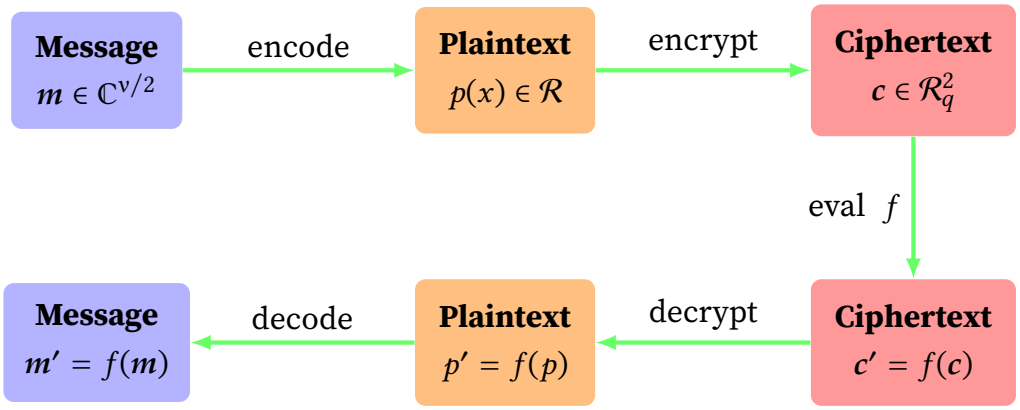
Σύστημα CKKS

Το σύστημα ομομορφικής κρυπτογράφησης 'Cheon–Kim–Kim–Song' (CKKS) [CKKS17] κατασκευάστηκε το 2017 με στόχο την επίτευξη προσεγγιστικών πράξεων σε κρυπτοκείμενα. Υποστηρίζει τις πράξεις της πρόσθεσης και του πολλαπλασιασμού κρυπτοκειμένων κατά προσέγγιση, χρησιμοποιώντας μια νέα διαδικασία ανακλιμάκωσης (rescaling) για τη διαχείριση του μεγέθους του αρχικού μηνύματος. Αυτή η διαδικασία μειώνει τον συντελεστή αναδίπλωσης του κρυπτοκειμένου, στρογγυλοποιώντας έτσι το αρχικό μήνυμα. Ως αποτέλεσμα, η αποκρυπτογράφηση εξάγει μία προσέγγιση της τιμής του μηνύματος με προκαθορισμένη ακρίβεια.

Το σύστημα CKKS έχει υλοποιηθεί από διάφορες βιβλιοθήκες ανοικτού κώδικα στη γλώσσα C++ (HEAAN, HElib, PALISADE, SEAL) και στη γλώσσα Go (Lattigo). Η βιβλιοθήκη SEAL, συγκεκριμένα, χρησιμοποιήθηκε σε αυτή την εργασία και περιγράφεται αργότερα.

Το σχήμα 2 παρέχει μία γενική επισκόπηση του συστήματος CKKS. Το σύστημα χρησιμοποιεί πολυωνυμικούς δακτυλίους διότι προσφέρουν μια αρκετά καλή ισορροπία μεταξύ ασφάλειας και απόδοσης. Αρχικά, λαμβάνει ένα διάνυσμα $\boldsymbol{m}$, στο οποίο αναμένεται να εκτελέσει κάποιον υπολογισμό. Στη συνέχεια, το κωδικοποιεί σε ένα πολυώνυμο $p(x)$, το οποίο και κρυπτογραφεί σε ένα διάνυσμα $\boldsymbol{c}$ που αποτελεί ένα ζεύγος πολυωνύμων. Έπειτα, εκτελεί σε αυτό κάποια ομομορφική πράξη f (όπως πρόσθεση, πολλαπλασιασμό, ή περιστροφή), η οποία παράγει ένα νέο διάνυσμα $\boldsymbol{c}'$. Τέλος, αποκρυπτογραφεί το $\boldsymbol{c}'$ στο πολυώνυμο p', το οποίο αποκωδικοποιεί για να δώσει το αποτέλεσμα ως το διάνυσμα $\boldsymbol{m}'$. [Huy20]

Σχήμα 2: Επισκόπηση του CKKS



Μαθηματικό Υπόβαθρο

Η ασφάλεια του συστήματος CKKS βασίζεται στη δυσκολία του προβλήματος 'Ring Learning With Errors' (RLWE) [LPR13], το οποίο εικάζεται ότι δεν μπορεί να επιλυθεί εύκολα ούτε σε κβαντικό υπολογιστή. Το πρόβλημα RLWE αποτελεί ειδίκευση του LWE [Reg05] με τη χρήση πολυωνυμικών δακτυλίων πάνω σε πεπερασμένα πεδία και δημιουργεί κλειδιά πολύ μικρότερου μεγέθους[3] από αυτό.

Αρχικά, επιλέγονται ομοιόμορφα δύο πολυώνυμα a και s από έναν δακτύλιο $\mathcal{R}_q = \mathbb{Z}_q[x]/f(x)$, όπου f είναι ένα αμείωτο πολυώνυμο βαθμού $\nu - 1$. Επίσης, λαμβάνεται ένα πολυώνυμο e βαθμού ν από την κατανομή λαθών $\mathcal{X}$, η οποία αποτελεί συνήθως μία διακριτή Gaussian κατανομή $\mathcal{X}_\sigma$ με τυπική απόκλιση σ. Η κατανομή $\mathcal{A}_{s,\mathcal{X}}$ στον χώρο $\mathcal{R}_q \times \mathcal{R}_q$ αποτελείται από πλειάδες (a, t) όπου $t = a \cdot s + e \in \mathcal{R}_q$. Το πρόβλημα αναφέρεται στην εύρεση του s δοθέντος ενός πολυωνυμικού πλήθους δειγμάτων (a, t) από την κατανομή $\mathcal{A}_{s,\mathcal{X}}$, κάτι που έχει αποδειχθεί πως είναι εξαιρετικά δύσκολο. [Cle+15 §2.1]

Συμβολισμοί

Οι ακόλουθοι συμβολισμοί χρησιμοποιούνται για την περιγραφή της υλοποίησης του συστήματος. [CKKS17 §2.1 §3.4, GHS15 §A.4]

$\langle v, w \rangle$ Το εσωτερικό γινόμενο δύο διανυσμάτων v και w.

$\overline{z}$ Ο συζυγής αριθμός του μιγαδικού z.

$\lfloor r \rceil$ Η στρογγυλοποίηση ενός πραγματικού αριθμού r στον κοντινότερό του ακέραιο (προς τα πάνω σε περίπτωση ισοβαθμίας).

S^n Το Καρτεσιανό γινόμενο ενός συνόλου S με τον εαυτό του εκτελεσμένο n φορές. (π.χ. $S^2 = S_1 \times S_2 = \{(s_1, s_2) \,|\, s_1 \in S_1, s_2 \in S_2\}$)

$g \circ f$ Η σύνθεση συνάρτησης $g(f(x))$ από δύο συναρτήσεις $g(x)$ και $f(x)$.

$s \leftarrow \mathcal{D}$ Η λήψη ενός τυχαίου δείγματος s από μία κατανομή $\mathcal{D}$.

$\mathcal{DG}(\sigma^2)$ Για έναν πραγματικό αριθμό $\sigma > 0$, η συνάρτηση λαμβάνει ένα διάνυσμα από την κανονική (Gaussian) κατανομή $\mathcal{N}(0, \sigma^2)^\nu$, με μέση τιμή 0 και απόκλιση σ^2, και το στρογγυλοποιεί στον κοντινότερο ακέραιο.

$\mathcal{HWT}(h)$ Για έναν ακέραιο αριθμό $h \geq 0$, η συνάρτηση λαμβάνει ένα διάνυσμα από το σύνολο $\{0, \pm 1\}^\nu$ εφόσον αυτό έχει βάρος Hamming (πλήθος τιμών ± 1) ίσο με h.

$\mathcal{ZO}(\rho)$ Για έναν πραγματικό αριθμό $\rho \in [0, 1]$, η συνάρτηση λαμβάνει ένα διάνυσμα από το σύνολο $\{0, \pm 1\}^\nu$ με πιθανότητα $\rho/2$ για τις τιμές ± 1 και $1 - \rho$ για την τιμή 0.

[3] Παραμένουν, όμως, μεγαλύτερα από τα κλειδιά παραδοσιακών αλγορίθμων όπως RSA και ECDH.

Υλοποίηση

Έστω L το υπολογιστικό βάθος του κυκλώματος και $\ell \in [0, \mathrm{L}]$ ένα επίπεδο εντός αυτού. Ο χώρος των μηνυμάτων ορίζεται από τον δακτύλιο $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^\nu + 1)$ που αναπαριστά το σύνολο των κυκλοτομικών πολυωνύμων βαθμού ν (δύναμη του 2), όπου q είναι ο συντελεστής αναδίπλωσης. Ορίζονται, επίσης, ένα πεδίο μιγαδικών αριθμών $\mathbb{H} = \{(z_j)_{j \in \mathbb{Z}_\mu^*} \mid z_j = \overline{z_{-j}}\} \subset \mathbb{C}^\nu$ και ένα πολλαπλασιαστικό υποσύνολο $\mathrm{T} \subset \mathbb{Z}_\mu^*$ τέτοιο ώστε $\mathbb{Z}_\mu^*/\mathrm{T} = \{\pm 1\}$, για κάποιον ακέραιο αριθμό μ. Επιπλέον, δηλώνεται ο ισομορφισμός $\varpi : (z_j)_{j \in \mathbb{Z}_\mu^*} \mapsto (z_j)_{j \in \mathrm{T}}$ και η εμφύτευση $\varsigma : \mathcal{R} \to \mathbb{C}^\nu$. Ακόμα, η n-οστή ρίζα της μονάδας απεικονίζεται ως $\zeta_n = e^{2\pi i/n}$ και ο παράγοντας ανακλιμάκωσης του μηνύματος πριν τη στρογγυλοποίηση ως $\Delta \geq 1$. [CKKS17 §2.2 §3.2–3.4]

Δημιουργία κλειδιών: $\mathtt{KeyGen}(1^\lambda)$

- Δοθείσας μίας παραμέτρου ασφαλείας λ, επιλέγονται οι αριθμοί $\mu = \mu(\lambda, q_\mathrm{L}) \in \mathbb{Z}$ (δύναμη του 2), $h = h(\lambda, q_\mathrm{L}) \in \mathbb{Z}$, $p = p(\lambda, q_\mathrm{L}) \in \mathbb{Z}$ και $\sigma = \sigma(\lambda, q_\mathrm{L}) \in \mathbb{R}$.
- Λαμβάνονται τα τυχαία δείγματα $s \leftarrow \mathcal{HWT}(h)$, $a \leftarrow \mathcal{R}_{q_\mathrm{L}}$, $e \leftarrow \mathcal{DG}(\sigma^2)$, $b \leftarrow -as + e \pmod{q_\mathrm{L}}$ και θέτονται το μυστικό κλειδί $sk \leftarrow (1, s)$ και το δημόσιο κλειδί $pk \leftarrow (b, a) \in \mathcal{R}_{q_\mathrm{L}}^2$.
- Λαμβάνονται, επίσης, τα $a' \leftarrow \mathcal{R}_{p \cdot q_\mathrm{L}}$, $e' \leftarrow \mathcal{DG}(\sigma^2)$, $b' \leftarrow -a's + e' + ps^2 \pmod{p \cdot q_\mathrm{L}}$ και τίθεται το κλειδί αποτίμησης $evk \leftarrow (b', a') \in \mathcal{R}_{p \cdot q_\mathrm{L}}^2$.

Κωδικοποίηση: $\mathtt{Ecd}(z; \Delta)$

Για ένα διάνυσμα ακεραίων του Gauss $z = (z_j)_{j \in \mathrm{T}} \in \mathbb{Z}[i]^{\nu/2}$ (μεγέθους $\nu/2$), η συνάρτηση επιστρέφει το πολυώνυμο $m(x) = \varsigma^{-1}(\lfloor \Delta \cdot \varpi^{-1}(z) \rceil_{\varsigma(\mathcal{R})}) \in \mathcal{R}$.

Αποκωδικοποίηση: $\mathtt{Dcd}(m; \Delta)$

Η κωδικοποίηση αντιστρέφεται εκτελώντας $\varpi \circ \varsigma(m)$ και η συνάρτηση επιστρέφει ένα διάνυσμα $z = (z_j)_{j \in \mathrm{T}} \in \mathbb{Z}[i]^{\nu/2}$ με $z_j = \lfloor \Delta^{-1} \cdot m(\zeta_\mu^j) \rceil$ για κάθε $j \in \mathrm{T}$.

Κρυπτογράφηση: $\mathtt{Enc}_{pk}(m)$

Η συνάρτηση λαμβάνει τα δείγματα $u \leftarrow \mathcal{ZO}(0.5)$, $e_0, e_1 \leftarrow \mathcal{DG}(\sigma^2)$ και επιστρέφει $\boldsymbol{c} = u \cdot pk + (m + e_0, e_1) \pmod{q_\mathrm{L}}$.

Αποκρυπτογράφηση: $\mathtt{Dec}_{sk}(\boldsymbol{c})$

Για $\boldsymbol{c} = (b, a)$, επιστρέφει $m = b + a \cdot sk \pmod{q_\ell}$.

Πρόσθεση: $\mathtt{Add}(\boldsymbol{c}_1, \boldsymbol{c}_2)$

Για $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \mathcal{R}_{q_\ell}^2$, επιστρέφει $\boldsymbol{c}_\mathtt{add} \leftarrow \boldsymbol{c}_1 + \boldsymbol{c}_2 \pmod{q_\ell}$.

Πολλαπλασιασμός: $\mathtt{Mult}_{evk}(\boldsymbol{c}_1, \boldsymbol{c}_2)$

Για $\boldsymbol{c}_1 = (b_1, a_1), \boldsymbol{c}_2 = (b_2, a_2) \in \mathcal{R}_{q_\ell}^2$, επιστρέφει $\boldsymbol{c}_\mathtt{mult} \leftarrow (d_0, d_1) + \lfloor p^{-1} \cdot d_2 \cdot evk \rceil \pmod{q_\ell}$, όπου $(d_0, d_1, d_2) = (b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2) \pmod{q_\ell}$.

Ανακλιμάκωση: $\mathtt{RS}_{\ell \to \ell'}(\boldsymbol{c})$

Για $\boldsymbol{c} \in \mathcal{R}_{q_\ell}^2$ και ένα επίπεδο $\ell' < \ell$, επιστρέφει $\boldsymbol{c}' \leftarrow \left\lfloor \frac{q_{\ell'}}{q_\ell} \boldsymbol{c} \right\rceil \pmod{q_{\ell'}}$.

Βιβλιοθήκη SEAL

Η 'Simple Encrypted Arithmetic Library' [SEAL] είναι μία βιβλιοθήκη ανοικτού κώδικα, υπό την άδεια MIT, με σκοπό την υλοποίηση συστημάτων ομομορφικής κρυπτογράφησης. Η ανάπτυξή της ξεκίνησε το 2015 από τη Microsoft Research και η πρώτη της έκδοση (3.1.0) δημοσιεύθηκε το 2018, ενώ αυτή τη στιγμή βρίσκεται στην έκδοση 3.7.2. Έχει αναπτυχθεί στη γλώσσα C++ δίχως εξάρτηση σε εξωτερικές βιβλιοθήκες, ώστε να μπορεί να εγκατασταθεί και να χρησιμοποιηθεί εύκολα ανεξαρτήτως πλατφόρμας. Είναι μάλιστα διαθέσιμη μέσω του διαχειριστή βιβλιοθηκών vcpkg και προσφέρεται επίσης από πακέτα για ορισμένες πλατφόρμες όπως Arch Linux (AUR), FreeBSD (Ports) και macOS (Homebrew).

Η βιβλιοθήκη επιτρέπει συγκεκριμένα την εκτέλεση προσθέσεων και πολλαπλασιασμών σε κρυπτογραφημένους ακέραιους, πραγματικούς, ή μιγαδικούς αριθμούς. Άλλες λειτουργίες, όπως η σύγκριση ή ταξινόμηση και η χρήση κανονικών εκφράσεων, δεν είναι εφικτό να πραγματοποιηθούν σε κρυπτογραφημένα δεδομένα μέσω ομομορφικού συστήματος. Επίσης, δεν είναι πάντα εύκολο να μετατραπεί ένας μη κρυπτογραφημένος υπολογισμός σε υπολογισμό πάνω σε κρυπτογραφημένα δεδομένα. Για παράδειγμα, δεν είναι δυνατή η χρήση αλμάτων και διακλαδώσεων όπως υποθετικών εκφράσεων, επαναλήψεων, και ετικετών.

Ασφάλεια

Σημειώνεται ότι τα ομομορφικά κρυπτοσυστήματα είναι εγγενώς εύπλαστα και δεν μπορούν να προσφέρουν πλήρη ασφάλεια απέναντι σε επιθέσεις γνωστών κρυπτοκειμένων. Η διαδικασία αποκρυπτογράφησης πρέπει να ελέγχει πάντα πρώτα την ορθότητα του κρυπτοκειμένου ώστε να μην επιτραπεί σε κάποιον επιτιθέμενο να λάβει πληροφορίες υποβάλλοντας δικό του κρυπτοκείμενο.

Επίσης, οι αποκρυπτογραφήσεις των παραγόμενων κρυπτοκειμένων δεν πρέπει να μοιράζονται με άλλους πέραν του κατόχου του μυστικού κλειδιού καθώς η κοινοποίησή τους ενδέχεται, σε ορισμένες περιπτώσεις, να οδηγήσει σε διαρροή του μυστικού κλειδιού. Οι Li και Micciancio έδειξαν πως μπορούν να εκτελεστούν κάποιες επιθέσεις στο σύστημα CKKS και μοιράστηκαν τα ευρήματά τους με τη SEAL και τις υπόλοιπες βιβλιοθήκες που υλοποιούν το σύστημα ώστε να διορθωθούν προτού δημοσιευθεί η έρευνά τους. [LM21]

Συστήματα

Η SEAL υλοποιεί δύο ξεχωριστά συστήματα ομομορφικής κρυπτογράφησης με πολύ διαφορετικές ιδιότητες μεταξύ τους.

BFV: Εκτελεί πράξεις αριθμητικής υπολοίπων σε κρυπτογραφημένους ακεραίους.

CKKS: Εκτελεί προσθέσεις και πολλαπλασιασμούς σε κρυπτογραφημένους πραγματικούς ή μιγαδικούς αριθμούς, αλλά παράγει μόνο προσεγγιστικά αποτελέσματα.

Το σύστημα BFV αποτελεί λοιπόν τη μόνη επιλογή όταν απαιτούνται ακριβείς τιμές. Από την άλλη, σε εφαρμογές όπως η άθροιση κρυπτογραφημένων πραγματικών αριθμών, η αποτίμηση μοντέλων μηχανικής μάθησης πάνω σε κρυπτογραφημένα δεδομένα, ή ο υπολογισμός αποστάσεων μεταξύ κρυπτογραφημένων τοποθεσιών, το σύστημα CKKS είναι μακράν η καλύτερη επιλογή.

Συμπίεση

Οι βιβλιοθήκες zstd και zlib μπορούν να χρησιμοποιηθούν για τη συμπίεση των κρυπτοκειμένων κατά τη σειριοποίηση. Η συμπίεση μπορεί να εφαρμοστεί σε οποιοδήποτε αντικείμενο της βιβλιοθήκης SEAL, από μυστικά και δημόσια κλειδιά, μέχρι μηνύματα και κρυπτοκείμενα, αλλά και οτιδήποτε άλλο μπορεί να σειριοποιηθεί. Αξίζει να σημειωθεί ότι, θεωρητικά, ο βαθμός συμπίεσης του μυστικού κλειδιού ενδέχεται να αποκαλύψει πληροφορίες σχετικές με αυτό.

Το σύστημα CKKS, ειδικότερα, επωφελείται αρκετά από τη συμπίεση διότι τα κρυπτοκείμενα αποτελούνται από πολλούς ακέραιους αριθμούς mod συγκεκριμένων πρώτων αριθμών, οι οποίοι μπορεί να είναι αρκετά μικροί (π.χ. 30 bits), ενώ σειριοποιούνται ως ακέραιοι με μέγεθος 64 bits. Έτσι, πολλά bytes του κρυπτοκειμένου είναι μηδενικά τα οποία απορρίπτονται κατά τη συμπίεση. Το σύστημα BFV επωφελείται συνήθως λιγότερο καθώς οι πρώτοι αριθμοί τείνουν να είναι μεγαλύτεροι, επομένως απομένουν λιγότερα μηδενικά bytes.

Αλγόριθμος Πολλαπλασιασμού

Ο πολλαπλασιασμός κρυπτογραφημένων πινάκων μεγέθους $d \times d$ μπορεί να εκτελεστεί όπως ακριβώς και ο γνωστός πολλαπλασιασμός μη κρυπτογραφημένων πινάκων, με πολυπλοκότητα τάξεως $O(d^3)$. Όμως, οι Jiang, Kim, Lauter και Song έδειξαν πως μπορεί να βελτιστοποιηθεί ο αλγόριθμος αναπαριστώντας τον κάθε πίνακα με ένα μόνο κρυπτοκείμενο (έναντι d^2), αξιοποιώντας την πράξη της περιστροφής. Η μέθοδος λειτουργεί με τρόπο SIMD και έχει πολυπλοκότητα τάξεως $O(d)$.

Συμβολισμοί

$\widehat{\mathrm{A}}$ Αναπαριστά την πράξη της μετατροπής (κατά σειρά) ενός πίνακα A διαστάσεων $m \times n$ σε διάνυσμα μεγέθους $m \cdot n$. Αντιστοιχεί στη συνάρτηση $\mathtt{vec}(\mathrm{A}^\top)$. [Har97 §16.2]

$\mathrm{A} \circ v$ Συμβολίζει το γινόμενο Hadamard [Mil07] μεταξύ ενός πίνακα A διαστάσεων $m \times n$ και ενός διανύσματος v μεγέθους n.

$[\![\mathrm{P}]\!]$ Οι αγκύλες Iverson [Knu92 §1] χρησιμοποιούνται για την αντιστοίχηση μίας λογικής έκφρασης P στον αριθμό 1 αν αυτή είναι αληθής ή 0 αν είναι ψευδής. Για λόγους σαφήνειας αξιοποιώ διπλές αγκύλες αντί για μονές.

Τετραγωνικοί Πίνακες

Έστω δύο κρυπτογραφημένοι πίνακες A, B μεγέθους $d \times d$ με $d = 3$ (σχήμα 3).

Ως $\mathtt{Rot}(\mathrm{C}; \vartheta)$ ορίζεται η συνάρτηση περιστροφής ενός ciphertext C κατά $|\vartheta|$ βήματα, αριστερόστροφα αν $\vartheta > 0$ ή δεξιόστροφα αν $\vartheta < 0$. [Jia+18 §2.2]

Επειδή το κάθε ciphertext αντιστοιχεί σε ένα διάνυσμα, μπορούμε να μετατρέψουμε τους πίνακες σε διανύσματα μεγέθους $n = d^2$, ώστε να ελαττώσουμε τον καθένα σε ένα μόνο ciphertext αντί για d. [Jia+18 §3.2]

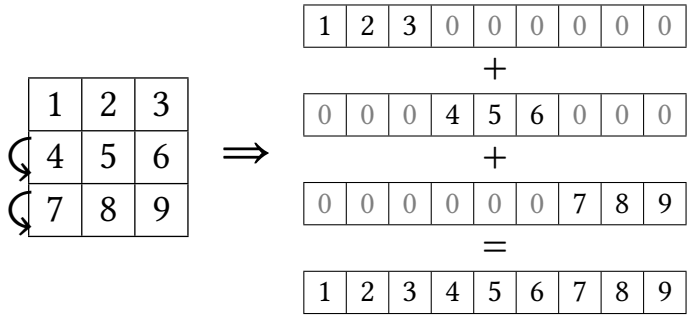
Η μετατροπή αυτή επιτυγχάνεται για την κάθε σειρά i του πίνακα A—και παρομοίως του B—εφαρμόζοντας $\mathtt{Rot}(\mathrm{A}; d \cdot i)$ και προσθέτοντας τα παραγόμενα διανύσματα όπως φαίνεται στο ακόλουθο σχήμα 4.

Σχήμα 3: Πίνακες 3×3

$$\mathrm{A} = \begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline \end{array}$$

$$\mathrm{B} = \begin{array}{|c|c|c|}\hline 9 & 8 & 7 \\ \hline 6 & 5 & 4 \\ \hline 3 & 2 & 1 \\ \hline \end{array}$$

Σχήμα 4: Μετατροπή πίνακα σε διάνυσμα



Για τις μεταθέσεις $\sigma, \tau, \phi^k, \psi^k$ ορίζουμε τους πίνακες $\mathrm{U}, \mathrm{Y}, \mathrm{V}^k, \mathrm{W}^k$ (μεγέθους $n \times n$) αντίστοιχα, σύμφωνα με τους παρακάτω τύπους. [Jia+18 §3.3.1]

$$\begin{aligned}
\sigma &: \mathrm{U}_{d\cdot i+j,l} &&= [\![l = d \cdot i + (i+j) \pmod d]\!] \\
\tau &: \mathrm{Y}_{d\cdot i+j,l} &&= [\![l = d \cdot (i+j) \pmod d + j]\!] \\
\phi^k &: \mathrm{V}^k_{d\cdot i+j,l} &&= [\![l = d \cdot i + (j+k) \pmod d]\!] \\
\psi^k &: \mathrm{W}^k_{d\cdot i+j,l} &&= [\![l = d \cdot (i+k) \pmod d + j]\!]
\end{aligned}$$

Έπειτα, μετασχηματίζουμε τους πίνακες $\mathrm{A} \to \mathrm{A}'$ και $\mathrm{B} \to \mathrm{B}'$ πολλαπλασιάζοντας τον κάθε πίνακα μετάθεσης με το αντίστοιχο διάνυσμα. [Jia+18 §2.3 §3.3.2]

- $\mathrm{A}'_0 = \mathrm{U} \circ \widehat{\mathrm{A}}$ (σχήμα 5)
- $\mathrm{B}'_0 = \mathrm{Y} \circ \widehat{\mathrm{B}}$ (σχήμα 6)
- $\forall k \in [1, d)\ \mathrm{A}'_k = \mathrm{V}^k \circ \mathrm{A}'_0$ (σχήμα 7)
- $\forall k \in [1, d)\ \mathrm{B}'_k = \mathrm{W}^k \circ \mathrm{B}'_0$ (σχήμα 8)

Τέλος, ορίζουμε το διάνυσμα $\widehat{\mathrm{AB}}$ ως το άθροισμα των πολλαπλασιασμών των επιμέρους διανυσμάτων και το μετατρέπουμε ξανά σε πίνακα για να λάβουμε το αποτέλεσμα (σχήμα 9).

$$\widehat{\mathrm{AB}} = \sum_{k=0}^{d} \mathrm{A}'_k \cdot \mathrm{B}'_k$$

Σχήμα 5: Γραμμικός μετασχηματισμός (σ)

1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0

×

1	2	3	4	5	6	7	8	9

=

1	2	3	5	6	4	9	7	8

Σχήμα 6: Γραμμικός μετασχηματισμός (τ)

1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0

×

9	8	7	6	5	4	3	2	1

=

9	5	1	6	2	7	3	8	4

Σχήμα 7: Γραμμικός μετασχηματισμός (ϕ^k)

(α′) $k = 1$

0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0	0

×

1	2	3	5	6	4	9	7	8

=

2	3	1	6	4	5	7	8	9

(β′) $k = 2$

0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0

×

1	2	3	5	6	4	9	7	8

=

3	1	2	4	5	6	8	9	7

Σχήμα 8: Γραμμικός μετασχηματισμός (ψ^k)

(α′) $k = 1$

0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0

×

9	5	1	6	2	7	3	8	4

=

6	2	7	3	8	4	9	5	1

(β′) $k = 2$

0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0

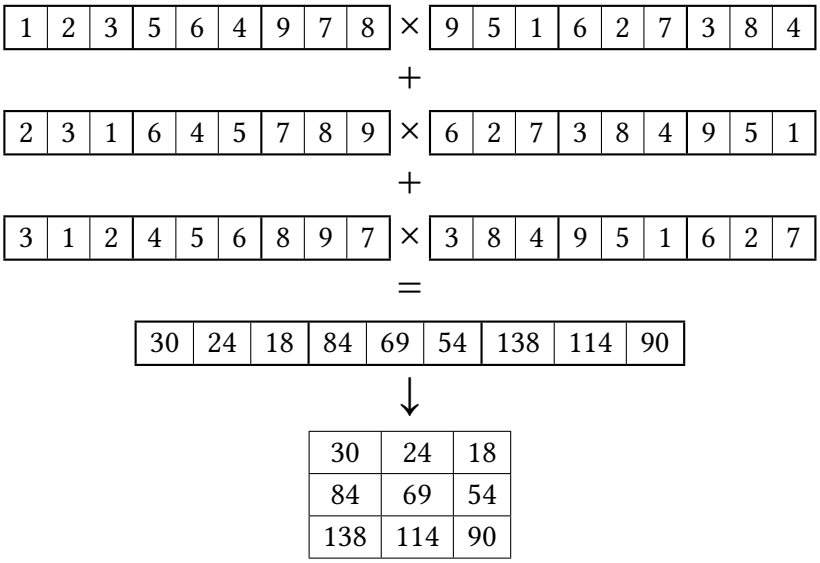
×

9	5	1	6	2	7	3	8	4

=

3	8	4	9	5	1	6	2	7

Σχήμα 9: Πολλαπλασιασμός διανυσμάτων



Ορθογώνιοι Πίνακες

Εάν ο πίνακας A δεν είναι τετραγωνικός, αλλά έχει μέγεθος $l \times d$ με $l < d$, μπορεί να μετατραπεί αφελώς σε τετραγωνικό γεμίζοντάς τον με μηδενικά και να εκτελεστεί ο πολλαπλασιασμός όπως ακριβώς φαίνεται παραπάνω, με πολυπλοκότητα της τάξεως $O(d)$. Σε περίπτωση, όμως, που ο αριθμός σειρών l είναι διαιρέτης του αριθμού στηλών d, υπάρχει δυνατότητα μείωσης της πολυπλοκότητας του αλγορίθμου στην τάξη $O(l)$. [Jia+18 §4.2]

Αυτό επιτυγχάνεται δημιουργώντας πρώτα έναν πίνακα $\underline{\mathrm{A}}$ ο οποίος περιέχει d/l κάθετα συνδεδεμένα αντίγραφα του αρχικού πίνακα A.

$$\underline{\mathrm{A}} = \left.\begin{bmatrix} \mathrm{A} \\ \cdots \\ \mathrm{A} \end{bmatrix}\right\} \frac{d}{l}$$

Έπειτα, εκτελούνται τα βήματα του παραπάνω αλγορίθμου, αντικαθιστώντας τον πίνακα A με τον $\underline{\mathrm{A}}$ και το μέγεθος d με το l. Το τελικό αποτέλεσμα δίνεται με ένα επιπλέον βήμα:

$$\widehat{\mathrm{AB}} = \sum_{k=0}^{\nu-1} \mathtt{Rot}(\widehat{\underline{\mathrm{A}}\mathrm{B}}; l \cdot d \cdot 2^k),\ \nu = \log_2\left(\frac{d}{l}\right)$$

Υλοποίηση

Ο κώδικας της παρούσας εργασίας υπάρχει στο GitLab υπό την άδεια MS-PL. Εκεί θα βρείτε οδηγίες ώστε να κάνετε compile και να δοκιμάσετε το πρόγραμμα σε περιβάλλον UN*X[4] .

Εκτός της βιβλιοθήκης SEAL, χρησιμοποίησα επίσης δύο single-header βιβλιοθήκες:

- cxx-prettyprint *(BSL-1.0)* για την παρουσίαση των πινάκων στον χρήστη.
- Quick Arg Parser *(MIT)* για την ερμηνεία των παραμέτρων του προγράμματος.

Αρχικά, έθεσα ορισμένες σταθερές όπως αυτές φαίνονται στον πίνακα 1.

`POLY_MOD_DEGREE` Ο βαθμός του πολυωνυμικού συντελεστή αναδίπλωσης.

`ENCODING_SCALE` Το όριο ακριβείας της κωδικοποίησης των κρυπτογραφημένων αριθμών.

`COEFF_MOD_BITS` Τα μεγέθη bits των πρώτων αριθμών που αξιοποιούνται στις πράξεις.

Πίνακας 1: Definitions

Name	Value
`POLY_MOD_DEGREE`	16834
`ENCODING_SCALE`	2^{40}
`COEFF_MOD_BITS`	[60 40 40 40 40 60]

Επίσης, όρισα τα type aliases του πίνακα 2 για ευκολία χρήσης.

Πίνακας 2: Type aliases

Type	Alias
`vector<double>`	`Mat1d`
`vector<vector<double>>`	`Mat2d`
`vector<vector<vector<double>>>`	`Mat3d`
`vector<Plaintext>`	`Mat1P`
`vector<vector<Plaintext>>`	`Mat2P`
`vector<Ciphertext>`	`Mat1C`

[4] Λειτουργεί και σε περιβάλλον Windows αλλά δεν υποστηρίζει τον MSVC compiler.

Utilities

Έπειτα, έγραψα ορισμένες συναρτήσεις για τη διευκόλυνση του χειρισμού τόσο των αριθμητικών πινάκων, όσο και των κρυπτογραφημένων.

Η συνάρτηση `matrix::random` λαμβάνει ως ορίσματα έναν άδειο αριθμητικό πίνακα δύο διαστάσεων *mat*, ένα κατώτατο όριο *lower* και ένα ανώτατο *upper* και γεμίζει τον πίνακα με ψευδοτυχαίους αριθμούς μεταξύ των ορίων $[lower, upper)$.

```
void matrix::random(Mat2d &mat, double lower, double upper) {
    assert(lower < upper);
    std::minstd_rand gen(std::random_device {}());
    std::uniform_real_distribution dist(lower, upper);
    for (auto i = mat.begin(); i != mat.end(); ++i)  // NOLINT
        for (auto j = i->begin(); j != i->end(); ++j) *j = dist(gen);
}
```
matrix/utils.cpp

Η συνάρτηση `matrix::diagonal` λαμβάνει ως ορίσματα έναν δισδιάστατο αριθμητικό πίνακα *mat*, και έναν αριθμό *n* και επιστρέφει τη n-οστή διαγώνιο του πίνακα εισόδου.

```
Mat1d matrix::diagonal(const Mat2d &mat, uint32_t n) {
    uint32_t k = 0, l = mat.size();
    Mat1d diagonal(l);
    for (uint32_t i = 0, j = n; i < l - n && j < l; ++i, ++j)
        diagonal[k++] = mat[i][j];
    for (uint32_t i = l - n, j = 0; i < l && j < n; ++i, ++j)
        diagonal[k++] = mat[i][j];
    return diagonal;
}
```
matrix/utils.cpp

Η συνάρτηση `matrix::diagonals` λαμβάνει ως όρισμα έναν δισδιάστατο αριθμητικό πίνακα *mat* και επιστρέφει έναν νέο πίνακα που περιέχει όλες τις διαγωνίους του αρχικού.

```
Mat2d matrix::diagonals(const Mat2d &mat) {
    uint32_t l = mat.size();
    Mat2d diagonals(l);
    for (uint32_t i = 0; i < l; ++i) diagonals[i] = diagonal(mat, i);
    return diagonals;
}
```
matrix/utils.cpp

Η συνάρτηση `matrix::multiply` υλοποιεί τον κλασσικό αλγόριθμο πολλαπλασιασμού τετραγωνικών πινάκων. Δέχεται ως ορίσματα δύο αριθμητικούς πίνακες *left*, *right* και επιστρέφει το γινόμενό τους σε έναν νέο πίνακα.

```
Mat2d matrix::multiply(const Mat2d &left, const Mat2d &right) {
    uint32_t l = left.size();
    Mat2d result(l, Mat1d(l));
    for (uint32_t i = 0; i < l; ++i)
        for (uint32_t j = 0; j < l; ++j)
            for (uint32_t k = 0; k < l; ++k)
                result[i][j] += left[i][k] * right[k][j];
    return result;
}
```
matrix/utils.cpp

Η συνάρτηση `matrix::crypto::encode`[5] λαμβάνει ως ορίσματα τον CKKS encoder *enc*, έναν δισδιάστατο αριθμητικό πίνακα *src*, και έναν άδειο plaintext πίνακα *dst* και κωδικοποιεί την κάθε γραμμή του πίνακα εισόδου σε ένα plaintext το οποίο αποθηκεύει στον πίνακα εξόδου.

```
    inline void encode(seal::CKKSEncoder &enc, Mat2d &src, Mat1P &dst) {
        assert(src.size() == dst.size());
        for (uint64_t i = 0, l = dst.size(); i < l; ++i)
            enc.encode(src[i], ENCODING_SCALE, dst[i]);
    }
```
matrix/crypto.hpp

Η συνάρτηση `matrix::crypto::encrypt`[5] λαμβάνει ως ορίσματα τον encryptor *enc*, έναν plaintext πίνακα *src*, και έναν ciphertext πίνακα *dst* και αποθηκεύει το ciphertext του κάθε στοιχείου του πίνακα εισόδου στον πίνακα εξόδου.

```
    inline void encrypt(seal::Encryptor &enc, Mat1P &src, Mat1C &dst) {
        assert(src.size() == dst.size());
        for (uint64_t i = 0, l = dst.size(); i < l; ++i)
            enc.encrypt(src[i], dst[i]);
    }
```
matrix/crypto.hpp

Η συνάρτηση `matrix::crypto::flatten` λαμβάνει ως ορίσματα τον evaluator *eval*, τα κλειδιά Galois *keys*, και έναν ciphertext πίνακα *mat* και επιστρέφει ένα μοναδικό ciphertext. Συγκεκριμένα, δημιουργεί από τον πίνακα εισόδου A μεγέθους l έναν νέο πίνακα R τέτοιον ώστε

$$\mathrm{R}_i = \begin{cases} \mathrm{A}_i & i = 0 \\ \mathrm{Rot}(\mathrm{A}_i, -i * l) & i \neq 0 \end{cases}$$

και προσθέτει όλα τα στοιχεία του σε ένα ciphertext, το οποίο επιστρέφει.

```
seal::Ciphertext matrix::crypto::flatten(
      seal::Evaluator &eval, const seal::GaloisKeys &keys, const Mat1C &mat) {
    seal::Ciphertext ct_result;
    auto l = static_cast<int32_t>(mat.size());

    Mat1C ct_dist(l);
    ct_dist[0] = mat[0];

    for (int32_t i = 1; i < l; ++i)
        eval.rotate_vector(mat[i], -i * l, keys, ct_dist[i]);

    eval.add_many(ct_dist, ct_result);

    return ct_result;
}
```
matrix/crypto.cpp

[5] Ως inline συνάρτηση, ο ορισμός της βρίσκεται μέσα στο namespace.

Permutations

Υλοποίησα τους αλγορίθμους των μεταθέσεων σ, τ, ϕ, ψ [Jia+18 §3.3.1] ως εξής.

```cpp
Mat2d matrix::permute::sigma(uint64_t d) {
    uint64_t n = d * d;
    Mat2d res(n, Mat1d(n));
    __PERMUTE(d * i + ((i + j) % d), res);
    return res;
}

Mat2d matrix::permute::tau(uint64_t d) {
    uint64_t n = d * d;
    Mat2d res(n, Mat1d(n));
    __PERMUTE(d * ((i + j) % d) + j, res);
    return res;
}

Mat2d matrix::permute::phi(uint64_t d, uint32_t k) {
    assert(k >= 1 && k < d);
    uint64_t n = d * d;
    Mat2d res(n, Mat1d(n));
    __PERMUTE(d * i + ((j + k) % d), res);
    return res;
}

Mat2d matrix::permute::psi(uint64_t d, uint32_t k) {
    assert(k >= 1 && k < d);
    uint64_t n = d * d;
    Mat2d res(n, Mat1d(n));
    __PERMUTE(d * ((i + k) % d) + j, res);
    return res;
}
```

matrix/permute.cpp

Οι συναρτήσεις `matrix::permute::sigma` και `matrix::permute::tau` λαμβάνουν ως είσοδο το μέγεθος d των δύο διαστάσεων των τετραγωνικών πίνακα των μεταθέσεων σ και τ αντίστοιχα. Οι πίνακες των μεταθέσεων ϕ και ψ, όμως, είναι τρισδιάστατοι· οπότε οι συναρτήσεις `matrix::permute::phi` και `matrix::permute::psi` δέχονται, εκτός του μεγέθους d, μία επιπλέον παράμετρο $k \in [1, d)$ η οποία ορίζει το συγκεκριμένο επίπεδο του τρισδιάστατου πίνακα που χρειαζόμαστε για να λάβουμε έναν δισδιάστατο.

Επειδή όλοι οι αλγόριθμοι δουλεύουν ουσιαστικά με τον ίδιο τρόπο, δημιούργησα το macro `__PERMUTE`, αλλάζοντας μόνο σε κάθε κλήση του την έκφραση *expr* που ορίζει την τιμή και τον πίνακα *result* όπου αποθηκεύει το αποτέλεσμα.

```cpp
#define __PERMUTE(expr, result)                                    \
    do {                                                           \
        for (uint32_t l = 0; l < n; ++l)                           \
            for (uint32_t i = 0; i < d; ++i)                       \
                for (uint32_t j = 0; j < d; ++j)                   \
                    (result)[d * i + j][l] = l == (expr) ? 1 : 1e-9; \
    } while (false)
```

matrix/permute.cpp

Το macro περιέχει πολλές διαφορετικές εκφράσεις οπότε το περιέκλεισα σε ένα `do ... while (false)` ώστε να μετατραπεί σε μοναδική έκφραση. Αυτό είναι μία καλή πρακτική αφού επιτρέπει πρώτον τη χρήση `;` μετά από αυτό και δεύτερον να συμπεριληφθεί αυτούσιο σε οποιαδήποτε κλήση συνάρτησης, υποθετική έκφραση, ή επανάληψη.

Αυτό που εκτελεί, λοιπόν, είναι μία επανάληψη τριών επιπέδων.

- Στο πρώτο επίπεδο θέτει τον ακέραιο $l \in [0, n)$ όπου $n = d^2$.
- Στο δεύτερο επίπεδο θέτει τον ακέραιο $i \in [0, d)$ για το αντίστοιχο d.
- Στο τρίτο επίπεδο θέτει τον ακέραιο $j \in [0, d)$ για το αντίστοιχο d.

Έπειτα, θέτει στο στοιχείο του πίνακα *result* που βρίσκεται στη σειρά $d * i + j$ και τη στήλη l την τιμή 1 αν η έκφραση *expr* ισοδυναμεί με τον ακέραιο l ή την τιμή 10^{-9} εάν αυτή διαφέρει. Η έκφραση *expr* αντιστοιχεί κάθε φορά στην επιθυμητή μετάθεση:

```
σ: d * i + ((i + j) % d)
τ: d * ((i + j) % d) + j
ϕ: d * i + ((j + k) % d)
ψ: d * ((i + k) % d) + j
```

Η τιμή του στοιχείου δεν μπορεί να τεθεί σε 0 καθώς αυτό δημιουργεί ένα «διάφανο» (transparent) ciphertext το οποίο δεν αποδέχεται η βιβλιοθήκη SEAL. Το πρόβλημα μπορεί ενδεχομένως να λυθεί με τέσσερις τρόπους:

α′ Κάνοντας με το χέρι compile τη βιβλιοθήκη θέτοντας την επιλογή `SEAL_THROW_ON_TRANSPARENT_CIPHERTEXT=OFF` κάτι που θεωρείται λανθασμένο και δεν είναι πρακτικό.

β′ Ελέγχοντας τα plaintext και ciphertext με τις μεθόδους τους `is_zero` και `is_transparent` αντίστοιχα, και αποφεύγοντας πράξεις σε αυτά εάν χρειάζεται. Ίσως είναι η πιο αποδοτική λύση, αλλά οδηγεί σε άλλα προβλήματα και περιπλέκει αρκετά τον κώδικα.

γ′ Αξιοποιώντας ένα `try ... catch` και τη μέθοδο `Encryptor::encrypt_zero` όταν εμφανίζεται σφάλμα διάφανου ciphertext. Αυτή η προσέγγιση επίσης περιπλέκει τον κώδικα και μοιάζει—κατά τη γνώμη μου—περισσότερο με «τσαπατσουλιά».

δ′ Χρησιμοποιώντας, αντί για 0, μία τιμή που βρίσκεται αρκετά κοντά σε αυτό, ώστε να μην είναι διάφανο το ciphertext, αλλά ούτε και να αλλοιωθούν τα αποτελέσματα. Τελικά, κατέληξα σε αυτόν τον τρόπο, θεωρώντας ότι η απλούστερη λύση είναι και η καλύτερη.

Linear Transformer

Για τον γραμμικό μετασχηματισμό [Jia+18 §2.3.3] δημιούργησα μία κλάση με δύο πεδία: τις παραμέτρους της κρυπτογράφησης *params_* και τα κλειδιά Galois *keys_*.

```
    explicit LinearTransformer(
          const seal::EncryptionParameters &params,
          const seal::GaloisKeys &keys):
        params_(params), keys_(keys) {}
```
linear_transformer.hpp

Υπερφόρτωσα τον τελεστή κλήσης ώστε να εκτελεί τον μετασχηματισμό. Δέχεται ως ορίσματα το *ciphertext* που θα μετασχηματίσει και έναν plaintext πίνακα *matrix* ο οποίος περιέχει τις διαγωνίους ενός πίνακα μετάθεσης.

Πρώτα, αρχικοποιεί έναν evaluator με τις παραμέτρους κρυπτογράφησης της κλάσης. Μετά, περιστρέφει δεξιόστροφα το *ciphertext* κατά l θέσεις (όπου l το μέγεθος του πίνακα *matrix*) και προσθέτει το αρχικό *ciphertext* συν το αποτέλεσμα της περιστροφής σε ένα νέο ciphertext. Επίσης, αποθηκεύει στη θέση 0 ενός ciphertext πίνακα το γινόμενο του πολλαπλασιασμού αυτού του νέου ciphertext με τη θέση 0 του plaintext πίνακα *matrix*.

Συνεχίζοντας, για κάθε αριθμό $i \in [1, l)$, περιστρέφει αριστερόστροφα κατά i θέσεις το νέο ciphertext που είχε δημιουργήσει πριν, πολλαπλασιάζει το αποτέλεσμα αυτό με τη θέση i του plaintext πίνακα, και αποθηκεύει το γινόμενο στη θέση i του ciphertext πίνακα. Τέλος, προσθέτει όλα τα στοιχεία του ciphertext πίνακα και επιστρέφει το αποτέλεσμα.

```
seal::Ciphertext LinearTransformer::operator()(
      const seal::Ciphertext &ciphertext, const Mat1P &matrix) {
    seal::SEALContext ctx(params_);
    seal::Evaluator eval(ctx);
    seal::Ciphertext ct_rot, ct_new, ct_result, temp_rot;

    auto l = static_cast<int32_t>(matrix.size());

    eval.rotate_vector(ciphertext, -l, keys_, ct_rot);
    eval.add(ciphertext, ct_rot, ct_new);

    Mat1C ct_dist(l);
    eval.multiply_plain(ct_new, matrix[0], ct_dist[0]);

    for (int32_t i = 1; i < l; ++i) {
        eval.rotate_vector(ct_new, i, keys_, temp_rot);
        eval.multiply_plain(temp_rot, matrix[i], ct_dist[i]);
    }

    eval.add_many(ct_dist, ct_result);

    return ct_result;
}
```
linear_transformer.cpp

Multiplier

Για τον πολλαπλασιασμό [Jia+18 §3.3.2] δημιούργησα πάλι μία κλάση με δύο πεδία: τις παραμέτρους της κρυπτογράφησης *params_* και τα κλειδιά Galois *keys_*.

```
    explicit Multiplier(
          const seal::EncryptionParameters &params,
          const seal::GaloisKeys &keys):
        params_(params), keys_(keys) {}
```
multiplier.hpp

Έφτιαξα μία static συνάρτηση με όνομα `stabilise_scale` η οποία δέχεται ως όρισμα ένα *ciphertext* και αλλάζει την κλίμακά του ώστε να είναι ίση με $\texttt{ENCODING\_SCALE}^2$. Αυτό είναι χρήσιμο καθώς, μετά τους μετασχηματισμούς, η κλίμακά του κάθε ciphertext βρίσκεται μεν κοντά σε αυτήν τη τιμή αλλά η μικρή διαφορά που υπάρχει αποτρέπει τις περαιτέρω πράξεις.

```
    static inline void stabilise_scale(seal::Ciphertext &ciphertext) {
        ciphertext.scale() = pow(ENCODING_SCALE, 2);
    }
```
multiplier.hpp

Ακόμα, υπερφόρτωσα τον τελεστή κλήσης ώστε να εκτελεί τον πολλαπλασιασμό. Δέχεται ως ορίσματα τους ciphertext πίνακες που πρέπει να πολλαπλασιάσει *ciphertexts*, το μέγεθος των πινάκων *size*, και τους plaintext πίνακες μετάθεσης *sigma*, *tau*, *phi*, *psi*.

```
seal::Ciphertext Multiplier::operator()(
      Operands &ciphertexts, uint64_t size, const Mat1P &sigma,
      const Mat1P &tau, const Mat2P &phi, const Mat2P &psi) {
```
multiplier.cpp

Πρώτα, αρχικοποιεί έναν evaluator με τις παραμέτρους κρυπτογράφησης της κλάσης και έναν Linear Transformer με τις ίδιες παραμέτρους καθώς και τα κλειδιά της κλάσης. Έπειτα, καλεί το `matrix::crypto::flatten` με τον evaluator, τα κλειδιά, και τους δύο ciphertext πίνακες αντίστοιχα, μετατρέποντας τον καθένα σε μονό ciphertext.

```
    seal::SEALContext ctx(params_);
    seal::Evaluator eval(ctx);
    LinearTransformer transform(params_, keys_);

    seal::Ciphertext product, temp;
    Mat1C ct_a(size), ct_b(size);

    auto ct1 = matrix::crypto::flatten(eval, keys_, ciphertexts.first);
    auto ct2 = matrix::crypto::flatten(eval, keys_, ciphertexts.second);
```
multiplier.cpp

`Operands = const pair<Mat1C, Mat1C>`

Αφού μετασχηματίσει το πρώτο ciphertext με το *sigma* και το δεύτερο με το *tau*, αποθηκεύει τα αντίστοιχα αποτελέσματα στη θέση 0 δύο προσωρινών ciphertext πινάκων. Ακόμα, πολλαπλασιάζει αυτές τις ciphertext τιμές, αποθηκεύει το γινόμενο σε νέο ciphertext, και μειώνει κατά έναν βαθμό το modulus του ciphertext αυτού.

```
    ct_a[0] = transform(ct1, sigma);
    ct_b[0] = transform(ct2, tau);

    eval.multiply(ct_a[0], ct_b[0], product);
    eval.mod_switch_to_next_inplace(product);
```
multiplier.cpp

Εν συνεχεία, για κάθε αριθμό $k \in [1, \text{size})$ αποθηκεύει στη θέση k των προαναφερόμενων πινάκων το αποτέλεσμα του μετασχηματισμού της θέσης 0 του αντίστοιχου πίνακα με τη γραμμή $k - 1$ του πίνακα *phi* για τον πρώτο ή *psi* για τον δεύτερο, μειώνει κατά έναν βαθμό τον συντελεστή του αποτελέσματος. Εδώ, μεταβάλλεται επίσης η κλίμακα των αποτελεσμάτων, η οποία σταθεροποιείται με τη χρήση του `stabilise_scale`. Επιπλέον, πολλαπλασιάζει τα δύο αυτά αποτελέσματα και προσθέτει το γινόμενο τους στο συνολικό, το οποίο επιστρέφει στο τέλος.

```
    for (uint64_t k = 1; k < size; ++k) {
        ct_a[k] = transform(ct_a[0], phi[k - 1]);
        eval.rescale_to_next_inplace(ct_a[k]);
        stabilise_scale(ct_a[k]);

        ct_b[k] = transform(ct_b[0], psi[k - 1]);
        eval.rescale_to_next_inplace(ct_b[k]);
        stabilise_scale(ct_b[k]);

        eval.multiply(ct_a[k], ct_b[k], temp);
        eval.add_inplace(product, temp);
    }

    return product;
```
multiplier.cpp

Main

Στο κύριο αρχείο του προγράμματος υπάρχει, αρχικά, ένα logging macro με ορίσματα το επίπεδο (*lvl*) και το μήνυμα (*msg*). Σε debug mode εμφανίζει επιπλέον το αρχείο προέλευσης και την γραμμή, αλλιώς μόνο το επίπεδο και το μήνυμα.

```cpp
#ifndef NDEBUG
#define LOG(lvl, msg) "[" #lvl "] (" __FILE__ ":" << __LINE__ << ") " << (msg)
#else
#define LOG(lvl, msg) "[" #lvl "] " << (msg)
#endif  // !NDEBUG
```
main.cpp

Έπειτα, η κλασσική main συνάρτηση ξεκινάει αναλύοντας τις παραμέτρους που δόθηκαν στο πρόγραμμα, όπως αυτές φαίνονται στον πίνακα 3. Θέτει τη μεταβλητή `size` και τη `size_sq` ως το τετράγωνό της. Εάν δόθηκε η παράμετρος `--verbose` τότε εμφανίζει τις τιμές όλων των παραμέτρων στον χρήστη.

```cpp
    Arguments args {{argc, argv}};

    const uint64_t size = args.size, size_sq = size * size;

    if (args.verbose) {
        std::cout << LOG(VERBOSE, "Size: ") << size << "x" << size
                  << ", Bounds: " << args.lower << "-" << args.upper
                  << std::endl;
    }
```
main.cpp

Πίνακας 3: Parameters

Παράμετρος	Ιδιότητα	Προεπιλογή
`--size` / `-s`	Θέτει το μέγεθος των πινάκων.	3
`--lower` / `-l`	Θέτει το κατώτατο όριο των τιμών.	0.1
`--upper` / `-u`	Θέτει το ανώτατο όριο των τιμών.	10.0
`--verbose` / `-v`	Εμφανίζει περισσότερα logs.	false

Εν συνεχεία, αρχικοποιεί τους πίνακες *A* και *B* με τυχαίες μεταξύ των δύο ορίων.

```cpp
    Mat2d A(size, Mat1d(size));
    Mat2d B(size, Mat1d(size));

    matrix::random(A, args.lower, args.upper);
    matrix::random(B, args.lower, args.upper);
```
main.cpp

Αν έχει γίνει compile σε debug mode, εμφανίζει τους πίνακες και το γινόμενό τους.

```cpp
#ifndef NDEBUG
    std::cout << LOG(DEBUG, "Matrix A: ") << A << std::endl;
    std::cout << LOG(DEBUG, "Matrix B: ") << B << std::endl;
    std::cout << LOG(DEBUG, "A * B: ") << matrix::multiply(A, B) << std::endl;
#endif  // !NDEBUG
```
main.cpp

Ακόμη, με την παράμετρο `--verbose` εμφανίζει τις παραμέτρους κρυπτογράφησης.

```
    if (args.verbose) {
        std::cout << LOG(VERBOSE, "Polynomial modulus degree: ")
                  << POLY_MOD_DEGREE << std::endl;
        std::cout << LOG(VERBOSE, "Coefficient modulus: ")
                  << std::vector COEFF_MOD_BITS << std::endl;
    }
```

main.cpp

Ο υπόλοιπος κώδικας βρίσκεται μέσα σε ένα **try ... catch** οπότε αν συμβεί οποιοδήποτε σφάλμα το εμφανίζει και κλείνει.

```
    } catch (const std::exception &exc) {
        std::cerr << LOG(ERROR, exc.what()) << std::endl;
        return 1;
    }
```

main.cpp

Αρχικοποιεί πρώτα τις παραμέτρους κρυπτογράφησης (*params*) και το *context* της SEAL χρησιμοποιώντας το σύστημα CKKS. Μετά παράγει το ζεύγος μυστικού (*sk*) και δημοσίου (*pk*) κλειδιού, καθώς και τα κλειδιά Galois (*gal_keys*) που χρησιμοποιούνται στους υπολογισμούς. Επιπλέον, αρχικοποιεί τον *encryptor*, τον *decryptor* και τον *encoder*.

```
        seal::EncryptionParameters params(seal::scheme_type::ckks);
        params.set_poly_modulus_degree(POLY_MOD_DEGREE);
        params.set_coeff_modulus(
              seal::CoeffModulus::Create(POLY_MOD_DEGREE, COEFF_MOD_BITS));
        seal::SEALContext context(params);

        seal::KeyGenerator keygen(context);
        seal::SecretKey sk = keygen.secret_key();
        seal::PublicKey pk;
        keygen.create_public_key(pk);
        seal::GaloisKeys gal_keys;
        keygen.create_galois_keys(gal_keys);

        seal::Encryptor encryptor(context, pk);
        seal::Decryptor decryptor(context, sk);
        seal::CKKSEncoder encoder(context);
```

main.cpp

Έπειτα, δημιουργεί τους δισδιάστατους πίνακες *sigma* και *tau* που περιέχουν τις διαγωνίους των αντίστοιχων πινάκων μετάθεσης. Παρόμοια, δημιουργεί τους τρισδιάστατους πίνακες *phi* και *psi* για κάθε επίπεδο.

```
        Mat2d sigma = matrix::diagonals(matrix::permute::sigma(size));
        Mat2d tau = matrix::diagonals(matrix::permute::tau(size));

        Mat3d phi(size - 1, Mat2d(size_sq, Mat1d(size_sq)));
        std::generate(phi.begin(), phi.end(), [size, k = 0]() mutable {
            return matrix::diagonals(matrix::permute::phi(size, ++k));
        });

        Mat3d psi(size - 1, Mat2d(size_sq, Mat1d(size_sq)));
        std::generate(psi.begin(), psi.end(), [size, k = 0]() mutable {
            return matrix::diagonals(matrix::permute::psi(size, ++k));
        });
```

main.cpp

Ύστερα, κωδικοποιεί τον δισδιάστατο αριθμητικό πίνακα *A* στον μονοδιάστατο plaintext πίνακα *A_pt* τον οποίο και κρυπτογραφεί στον ciphertext πίνακα *A_ct*. Αντίστοιχα, κωδικοποιεί τον πίνακα *B* στον πίνακα *B_pt* και τον κρυπτογραφεί στον πίνακα *B_ct*.

```
        Mat1P A_pt(size);
        matrix::crypto::encode(encoder, A, A_pt);
        Mat1C A_ct(size);
        matrix::crypto::encrypt(encryptor, A_pt, A_ct);

        Mat1P B_pt(size);
        matrix::crypto::encode(encoder, B, B_pt);
        Mat1C B_ct(size);
        matrix::crypto::encrypt(encryptor, B_pt, B_ct);
```
main.cpp

Στη συνέχεια, πολλαπλασιάζει τους πίνακες *A_ct* και *B_ct* χρησιμοποιώντας τον Multiplier *multiply* και αποθηκεύει το γινόμενο στο ciphertext *AB_ct*. Επίσης, αποκρυπτογραφεί το ciphertext στο plaintext *AB_pt* το οποίο και αποκωδικοποιεί στον (μονοδιάστατο) αριθμητικό πίνακα *AB_res_1d*.

```
        Multiplier multiply(params, gal_keys);
        seal::Ciphertext AB_ct =
              multiply({A_ct, B_ct}, size, sigma_pt, tau_pt, phi_pt, psi_pt);

        seal::Plaintext AB_pt;
        decryptor.decrypt(AB_ct, AB_pt);

        Mat1d AB_res_1d(size_sq);
        encoder.decode(AB_pt, AB_res_1d);
```
main.cpp

Τέλος, μετατρέπει τον πίνακα σε δισδιάστατο (*AB_res_2d*) και τον εμφανίζει στον χρήστη.

```
        Mat2d AB_res_2d(size, Mat1d(size));
        for (uint64_t i = 0; i < size_sq; ++i)
            AB_res_2d[i / size][i % size] = AB_res_1d[i];

        std::cout << LOG(INFO, "CT(A) * CT(B): ") << AB_res_2d << std::endl;
```
main.cpp

Αποτελέσματα

Εκτελώντας το πρόγραμμα, λαμβάνουμε ένα αποτέλεσμα όπως φαίνεται στο σχήμα 10 παρακάτω. Φυσικά, καθώς το πρόγραμμα παράγει τους πίνακες ψευδοτυχαία, οι τιμές τους σε κάθε εκτέλεση θα είναι διαφορετικές.

Σχήμα 10: Αποτέλεσμα εκτέλεσης

```
[VERBOSE] (src/main.cpp:43) Size: 3x3, Bounds: 0.1-10
[DEBUG] (src/main.cpp:55) Matrix A:
↪ [[1.12801, 0.687072, 4.33984], [1.98936, 8.10535, 1.26065], [9.7498, 9.47494, 7.94238]]
[DEBUG] (src/main.cpp:56) Matrix B:
↪ [[0.958044, 2.10459, 4.75032], [3.69101, 9.18964, 4.16116], [3.22228, 8.54748, 3.18408]]
[DEBUG] (src/main.cpp:57) A * B:
↪ [[17.6009, 45.7826, 22.0359], [35.885, 89.4474, 47.1917], [69.9055, 175.478, 111.031]]
[VERBOSE] (src/main.cpp:61) Polynomial modulus degree: 16384
[VERBOSE] (src/main.cpp:63) Coefficient modulus: [60, 40, 40, 40, 40, 60]
[INFO] (src/main.cpp:138) CT(A) * CT(B):
↪ [[17.6009, 45.7828, 22.0359], [35.885, 89.4476, 47.1918], [69.9056, 175.478, 111.031]]
```

Παρατηρούμε ότι το γινόμενο των κρυπτογραφημένων πινάκων $\mathrm{CT(A)} * \mathrm{CT(B)}$ έχει μόνο μία αμελητέα απόκλιση από το γινόμενο των αρχικών πινάκων $\mathrm{A} * \mathrm{B}$.

Σημειώνεται ότι το μέγεθος των πινάκων και τα όρια των τυχαίων τιμών μπορούν να αλλάξουν θέτοντας τις κατάλληλες παραμέτρους του πίνακα 3. Στην προκειμένη περίπτωση χρησιμοποίησα την παράμετρο `--verbose` για να εμφανίσει ορισμένα επιπλέον στοιχεία σχετικά με τους πίνακες και την κρυπτογράφηση.

Έλεγχος Επιδόσεων

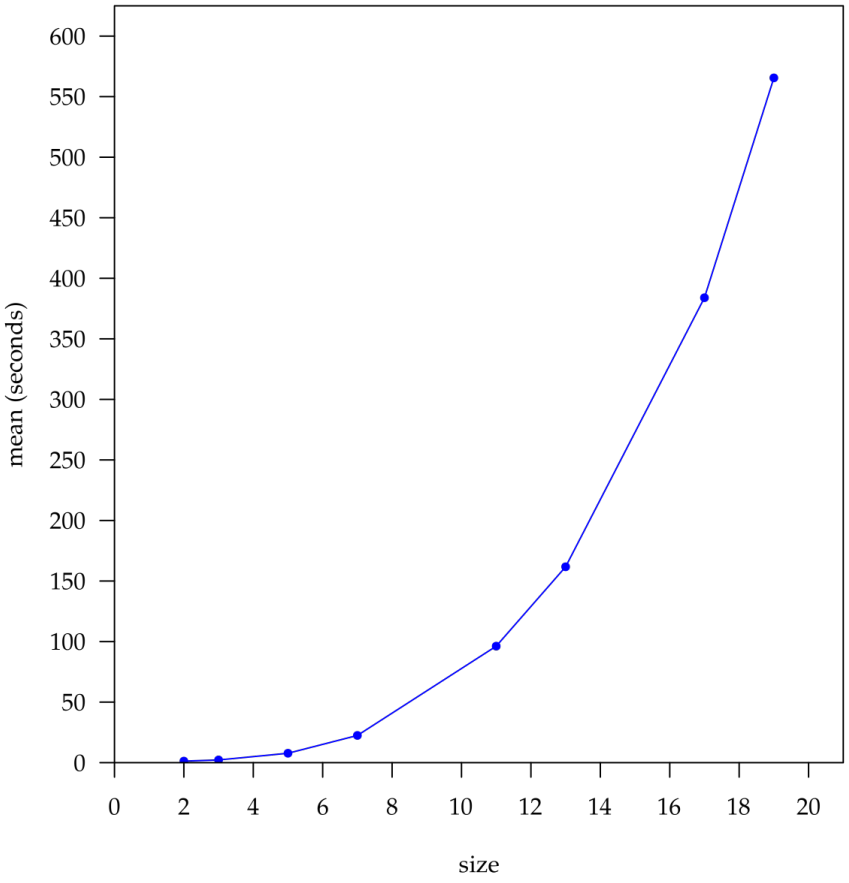
Ο πίνακας 4 περιέχει τον μέσο όρο, την τυπική απόκλιση, και τη μικρότερη και μεγαλύτερη τιμή του χρόνου εκτέλεσης του προγράμματος (σε δευτερόλεπτα) για διάφορα μεγέθη πινάκων εισόδου. Για την εκτέλεση του ελέγχου επιδόσεων χρησιμοποιήθηκε το εργαλείο hyperfine. Το πρόγραμμα έγινε compile με χρήση του clang compiler (έκδοση 13.0) και οι δοκιμές εκτελέστηκαν σε λογισμικό Arch Linux (έκδοση πυρήνα 5.10) με οκταπύρηνο επεξεργαστή AMD Ryzen 7 3700U και μνήμη 16 gigabyte.

Παρατηρείται ότι ο μέσος χρόνος εκτέλεσης του προγράμματος αυξάνεται εκθετικά όσο μεγαλώνει το μέγεθος των πινάκων εισόδου (σχήμα 11) και υπάρχει ισχυρή γραμμική συσχέτιση μεταξύ του μεγέθους και του χρόνου, αφού ο συντελεστής συσχέτισης του Pearson ($\varrho = 0.9223656$) βρίσκεται πολύ κοντά στη μονάδα. Αντιθέτως, ο πολλαπλασιασμός μη κρυπτογραφημένων πινάκων χρειάζεται μόνο ελάχιστα χιλιοστά του δευτερολέπτου και ο χρόνος εκτέλεσής του παραμένει σχεδόν σταθερός για μικρά μεγέθη πινάκων εισόδου ενώ αυξάνεται μόνο για πολύ μεγαλύτερα μεγέθη.

Πίνακας 4: Benchmarks

	Mean	Std. Dev.	Min	Max
2×2	1.275	0.008	1.266	1.285
3×3	2.252	0.068	2.211	2.372
5×5	7.782	0.092	7.707	7.931
7×7	22.490	0.163	22.216	22.649
11×11	96.216	0.190	95.966	96.425
13×13	161.766	0.164	161.494	161.936
17×17	383.946	0.264	383.638	384.229
19×19	565.494	0.681	564.470	566.311

Σχήμα 11: Χρόνοι εκτέλεσης



Βελτιώσεις

Η διαδικασία πολλαπλασιασμού των κρυπτογραφημένων πινάκων είναι κατά μεγάλο βαθμό βραδύτερη από τον κλασσικό πολλαπλασιασμό των αρχικών πινάκων. Ο μεγάλος χρόνος εκτέλεσης των ομομορφικών υπολογισμών είναι μεν αναμενόμενος, αλλά ενδέχεται ίσως να μειωθεί παραλληλοποιώντας ορισμένες επαναλήψεις ή άλλα κομμάτια του κώδικα (π.χ. με τη χρήση OpenMP [DM98]).

Ένα ακόμη κομμάτι του προγράμματος που χρήζει βελτίωσης είναι η δημιουργία των πινάκων μετάθεσης. Ο αλγόριθμος που δημιουργεί αυτούς τους πίνακες έχει, από μόνος του, πολυπλοκότητα τάξεως $O(d^4)$, η οποία αυξάνεται ακόμα περισσότερο όταν λαμβάνονται οι διαγώνιοι. Θα μπορούσαν να δημιουργούνται απευθείας οι διαγώνιοι (εφόσον οι αρχικοί πίνακες δε χρησιμεύουν αλλού) και, πιθανών, αυτό να γίνεται με πιο αποδοτικό τρόπο.

Επιπλέον, οι πίνακες μεταθέσεων εξαρτώνται μόνο από τα μεγέθη των πινάκων εισόδου και όχι από τα περιεχόμενα. Επομένως, υπάρχει η δυνατότητα να σειριοποιηθούν εκ των προτέρων οι plaintext πίνακες των διαγωνίων και να αποθηκευθούν σε αρχεία (ξεχωριστά για κάθε μέγεθος), ώστε να διαβάζονται από εκεί αντί να υπολογίζονται εκ νέου για κάθε καινούρια είσοδο. Φυσικά, αυτό ενδέχεται να δημιουργήσει πρόβλημα εάν ο χώρος αποθήκευσης είναι περιορισμένος.

Τέλος, σε περίπτωση που το μέγεθος των πινάκων μετάθεσης αποτελεί δύναμη του 2, τότε μπορεί να χρησιμοποιηθεί η «υβριδική» τεχνική του πλαισίου ‘GAZELLE’ [JVC18 §5.2] για τον πολλαπλασιασμό τους με το κρυπτοκείμενο. Αυτή συνδυάζει την τεχνική των διαγωνίων με τον αφελή πολλαπλασιασμό, μειώνοντας το μέγεθος των πινάκων μετάθεσης και τον αριθμό των περιστροφών που απαιτούνται για την εκτέλεση του πολλαπλασιασμού.

Επίλογος

Κλείνοντας, ας δούμε μερικές πιθανές εφαρμογές της ομομορφικής κρυπτογράφησης [Ara20], καθώς και τα συμπεράσματα που προέκυψαν κατά την έρευνα και υλοποίηση της παρούσας εργασίας.

Εφαρμογές

Ένα παράδειγμα που έδωσε ο Gentry κατά τη δημιουργία του πρώτου συστήματος FHE [Gen09] ήταν η υλοποίηση μηχανής αναζήτησης η οποία να μπορεί να επιστρέφει αποτελέσματα χωρίς να γνωρίζει το ερώτημα που της δόθηκε από τον χρήστη. Περαιτέρω, έχουν βρεθεί οι ακόλουθες πρακτικές εφαρμογές:

Ασφάλεια Ηλεκτρονικών Ψηφοφοριών

Ερευνάται η χρήση της ομομορφικής κρυπτογράφησης για τη διοργάνωση εκλογών ηλεκτρονικά, με την ίδια ασφάλεια και διαφάνεια που προσφέρουν οι παραδοσιακές εκλογές. Για παράδειγμα, το κρυπτοσύστημα Paillier [Pai99], το οποίο υποστηρίζει κρυπτογραφημένες προσθέσεις, φαίνεται κατάλληλο για την επίτευξη αμερόληπτων ηλεκτρονικών ψηφοφοριών. Αυτή η τεχνολογία μπορεί να προστατεύσει τις ψήφους από παραποίηση, αλλά και να επιτρέψει την επαλήθευση τους από εξουσιοδοτημένα τρίτα μέρη.

Ακεραιότητα στο Υπολογιστικό Νέφος

Με την ομομορφική κρυπτογράφηση μπορεί να χρησιμοποιηθεί από έναν ιδιώτη ή μία εταιρία κάποια υπηρεσία υπολογιστικού νέφος για την αποθήκευση προσωπικών ή ιδιόκτητων δεδομένων, ακόμη κι αν ο πάροχός της δεν είναι άξιος πλήρους εμπιστοσύνης. Έτσι, διατηρείται η δυνατότητα υπολογισμού και αναζήτησης σε κρυπτογραφημένες πληροφορίες μέσω της υπηρεσίας και μπορεί να αποκρυπτογραφηθεί το αποτέλεσμα αργότερα τοπικά, χωρίς να τεθεί σε κίνδυνο η ακεραιότητα των δεδομένων.

Ανάλυση Απόρρητων Δεδομένων

Η ομομορφική κρυπτογράφηση μπορεί να επιτρέψει τον εξωπορισμό (outsourcing) κρυπτογραφημένων δεδομένων για σκοπούς έρευνας και κοινής χρήσης τους, προστατεύοντας παράλληλα το απόρρητο των κατόχων αυτών των δεδομένων. Μερικά παραδείγματα αυτής της χρήσης είναι η προγνωστική ανάλυση ιατρικών δεδομένων χωρίς να διακυβεύεται το απόρρητο των ασθενών, η εξατομίκευση των διαφημίσεων διατηρώντας ταυτόχρονα το απόρρητο των χρηστών, η αξιολόγηση της δυνατότητας εξόφλησης δανείου από έναν πελάτη κάποιας χρηματοοικονομικής εταιρίας, καθώς και η ιατροδικαστική αναγνώριση εικόνων.

Συμπεράσματα

Στον σημερινό κόσμο που έχει ως επίκεντρο το διαδίκτυο, η ασφάλεια των δεδομένων διαδραματίζει πιο σημαντικό ρόλο από ποτέ. Για εξαιρετικά ευαίσθητα συστήματα όπως το διαδικτυακό εμπόριο (e-commerce) και η ηλεκτρονική τραπεζική (e-banking), είναι απαραίτητο να προστατεύονται οι λογαριασμοί και τα περιουσιακά στοιχεία των χρηστών από κακόβουλα τρίτα μέρη. Όπως φαίνεται, λοιπόν, η ομομορφική κρυπτογράφηση θα αποτελέσει ένα πολύτιμο εργαλείο στο—όχι πολύ μακρινό—μέλλον της πληροφορικής επιστήμης, καθώς προσφέρει τη δυνατότητα σε υπηρεσίες που χειρίζονται ευαίσθητα δεδομένα να τα αποθηκεύουν κρυπτογραφημένα σε κάποιον πάροχο υπολογιστικού νέφους και να τα επεξεργάζονται απευθείας σε αυτό χωρίς να διακινδυνεύουν την ασφάλειά τους.

Προς το παρών, όμως, υπάρχουν δύο παράγοντες που αποθαρρύνουν την υιοθέτηση συστημάτων ομομορφικής κρυπτογράφησης: η περιορισμένη ταχύτητα και η δυσκολία στη χρήση. Το πρόβλημα της ταχύτητας μπορεί να λυθεί (τουλάχιστον εν μέρει) ενσωματώνοντας ορισμένες οδηγίες απαραίτητες για τη λειτουργία του ομομορφικού συστήματος στον επεξεργαστή, όπως στην περίπτωση του AES και άλλων κρυπτογραφικών αλγορίθμων, ή σε κάποιο FPGA. Ήδη, αναπτύσσεται η βιβλιοθήκη HEXL η οποία αξιοποιεί το σύνολο εντολών AVX-512, που διαθέτουν ορισμένοι σύγχρονοι επεξεργαστές της εταιρίας Intel, για την επιτάχυνση της ομομορφικής κρυπτογράφησης σε χαμηλό επίπεδο. Η χρήση της ομομορφικής κρυπτογράφησης από εταιρίες που δεν είναι γνώριμες με αυτήν μπορεί να διευκολυνθεί από τον πάροχο του υπολογιστικού νέφους αφομοιώνοντας το κρυπτοσύστημα στην υπηρεσία του και προσφέροντας μια ιστοσελίδα για τη διαχείρισή του και κάποιο REST API ή RPC για τον προγραμματιστή.

Η παρούσα εργασία άγγιξε μόνο την επιφάνεια των υπολογισμών που μπορούν να εκτελεσθούν με τη χρήση συστημάτων ομομορφικής κρυπτογράφησης, έχοντας ως στόχο την επίδειξη και υλοποίηση ενός προγράμματος πολλαπλασιασμού κρυπτογραφημένων πινάκων. Θα μπορούσε να αναπτυχθεί μία πιο ολοκληρωμένη βιβλιοθήκη για τη δημιουργία μοντέλων λογιστικής παλινδρόμησης και μηχανικής μάθησης, αξιοποιώντας επίσης τεχνικές του πλαισίου ‘GAZELLE’ [JVC18].

Ορολογίες

ciphertext (κρυπτοκείμενο)
: Το κωδικοποιημένο κείμενο που παράγει ένας αλγόριθμος κρυπτογράφησης.

circuit (κύκλωμα)
: Ένα υπολογιστικό μοντέλο όπου οι τιμές εισόδου προχωρούν μέσα από μία σειρά πυλών, καθεμία εκ των οποίων υπολογίζει κάποια συνάρτηση.

cryptosystem (κρυπτοσύστημα)
: Ένα σύνολο αλγορίθμων που χρησιμοποιούνται για την υλοποίηση κάποιας μεθόδου κρυπτογράφησης και αποκρυπτογράφησης.

cyclotomic (κυκλοτομικό)
: Κάθε πολυώνυμο του οποίου οι μιγαδικές ρίζες είναι πρωταρχικές ρίζες της μονάδας.

eigenvector (ιδιοδιάνυσμα)
: Ένα διάνυσμα που διατηρεί την κατεύθυνσή του μετά από μετασχηματισμό του.

embedding (εμφύτευση)
: Μια αναπαράσταση κάποιας αλγεβρικής δομής σε έναν συγκεκριμένο χώρο η οποία διατηρεί τις ιδιότητές της.

encryption (κρυπτογράφηση)
: Η διαδικασία κωδικοποίησης ενός μηνύματος σε μία ακατανόητη μορφή.

field (πεδίο)
: Ένα είδος αλγεβρικού συνόλου που υποστηρίζει και τις τέσσερις βασικές πράξεις.

Galois key (κλειδί Galois)
: Το κλειδί ενός κρυπτογραφικού αλγορίθμου που χρησιμοποιεί τη μέθοδο GCM.

Gaussian integer (ακέραιος του Gauss)
: Μιγαδικός αριθμός με ακέραιο πραγματικό και φανταστικό μέρος.

homomorphism (ομομορφισμός)
: Η αντιστοίχηση ενός συνόλου δεδομένων σε ένα άλλο διατηρώντας παράλληλα τις σχέσεις μεταξύ των στοιχείων και στα δύο σύνολα.

isomorphism (ισομορφισμός)
: Ομομορφισμός όπου ισχύει και το αντίστροφο.

lattice (δικτυωτό)
: Μία περιοδική διάταξη σημείων του πραγματικού διανυσματικού χώρου.

linear transformation (γραμμικός μετασχηματισμός)
: Μία συνάρτηση μεταξύ δύο διανυσματικών χώρων η οποία διατηρεί τις πράξεις της διανυσματικής πρόσθεσης και του βαθμωτού πολλαπλασιασμού.

linearisation (γραμμικοποίηση)
: Η διαδικασία εύρεσης μίας ευθείας γραμμής (γνωστή ως γραμμική προσέγγιση) που ταιριάζει σε κάποια τοποθεσία μίας συνάρτησης.

logistic regression (λογιστική παλινδρόμηση)
: Ένα στατιστικό μοντέλο που χρησιμοποιείται για την εκτίμηση της πιθανότητας ενός συμβάντος ή χαρακτηριστικού με βάση συγκεκριμένα δεδομένα.

machine learning (μηχανική μάθηση)
: Ένα πεδίο μελέτης που δίνει στους υπολογιστές την ικανότητα να «μαθαίνουν» κάτι, χωρίς αυτό να έχει ρητά προγραμματιστεί.

malleable (εύπλαστο)
: Ένα κρυπτοσύστημα που επιτρέπει τον μετασχηματισμό ενός κρυπτοκειμένου σε άλλο, η αποκρυπτογράφηση του οποίου σχετίζεται με το αρχικό κείμενο.

matrix (πίνακας)
: Μία ορθογώνια διάταξη αριθμών σε σειρές και στήλες.

modulus (συντελεστής αναδίπλωσης)
: Το όριο των ακεραίων τιμών ενός συστήματος αριθμητικής υπολοίπων.

permutation (μετάθεση)
: Η τοποθέτηση των στοιχείων ενός συνόλου με συγκεκριμένη σειρά.

plaintext (απλό κείμενο)
: Τα δεδομένα που εισάγονται σε έναν αλγόριθμο κρυπτογράφησης.

polynomial (πολυώνυμο)
: Μία έκφραση κατασκευασμένη από μεταβλητές και σταθερές χρησιμοποιώντας μόνο στοιχειώδεις μαθηματικές πράξεις.

ring (δακτύλιος)
: Ένα είδος αλγεβρικού συνόλου που υποστηρίζει προσθέσεις και πολλαπλασιασμούς.

serialisation (σειριοποίηση)
: Η διαδικασία μετατροπής μίας δομής δεδομένων σε αποθηκεύσιμη μορφή.

tuple (πλειάδα)
: Μία πεπερασμένη ακολουθία στοιχείων.

vector (διάνυσμα)
: Ένα διατεταγμένο σύνολο αριθμών.

Συντομογραφίες

ACM Association for Computing Machinery
AES Advanced Encryption Standard
AMD Advanced Micro Devices
API Application Programming Interface
AVX Advanced Vector Extensions
BFV Brakerski/Fan–Vercauteren
BGN Boneh–Goh–Nissim
BGV Brakerski–Gentry–Vercauteren
BLLN Bos–Lauter–Loftus–Naehrig
BSL-1.0 Boost Software License 1.0
CKKS Cheon–Kim–Kim–Song
DNF Disjunctive Normal Form
ECDH Elliptic-Curve Diffie–Hellman
EPFL Federal Institute of Technology Lausanne (École Polytechnique Fédérale de Lausanne)
FHE Fully Homomorphic Encryption
FPGA Field-Programmable Gate Array
GCM Galois/Counter Mode
GSW Gentry–Sahai–Waters
HEAAN Homomorphic Encryption for Arithmetic of Approximate Numbers
HEXL Homomorphic Encryption Acceleration Library
IACR International Association for Cryptologic Research
IBE Identity-Based Encryption
IEEE Institute of Electrical and Electronics Engineers
IMA Institute of Mathematics & its Applications
IND-CPA Indistinguishability under Chosen Plaintext Attack
LDS Laboratory for Data Security
LTV López–Tromer–Vaikuntanathan
LWE Learning With Errors
MIT Massachusetts Institute of Technology
MS-PL Microsoft Public License
NTRU N-th degree Truncated polynomial Ring Units
PHE Partially Homomorphic Encryption
REST Representational State Transfer
RLWE Ring Learning With Errors
RPC Remote Procedure Call
RSA Rivest–Shamir–Adleman
SEAL Simple Encrypted Arithmetic Library
SHE/SWHE Somewhat Homomorphic Encryption
SIMD Single Instruction, Multiple Data

Βιβλιογραφία

[ABD16] Martin Albrecht, Shi Bai & Léo Ducas. 'A Subfield Lattice Attack on Overstretched NTRU Assumptions: Cryptanalysis of Some FHE and Graded Encoding Schemes'. In: *International Conference on Advances in Cryptology*. 36. IACR. 2016, pp. 153–178. DOI: 10.1007/978-3-662-53018-4_6.

[Aca+18] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac & Mauro Conti. 'A Survey on Homomorphic Encryption Schemes: Theory and Implementation'. In: *ACM Computing Surveys* 51.4. ACM, 2018, 35 pp. DOI: 10.1145/3214303.

[Alb+18] Martin Albrecht, Melissa Chase, Hao Chen et al. *Homomorphic Encryption Security Standard*. Tech. rep. Release 1.1. HomomorphicEncryption.org, 21st Nov. 2018.

[Ara20] Anastasios Arampatzis. *Homomorphic Encryption: What Is It and How Is It Used*. Venafi. 22nd Jan. 2020. URL: https://www.venafi.com/blog/homomorphic-encryption-what-it-and-how-it-used.

[Ben20] Ayoub Benaissa. *Introduction to Homomorphic Encryption*. 2nd Jan. 2020. URL: https://www.ayoub-benaissa.com/blog/introduction-to-homomorphic-encryption/.

[BFV12] Junfeng Fan & Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Jan. 2012. IACR: 2012/144.

[BGN05] Dan Boneh, Eu-Jin Goh & Kobbi Nissim. 'Evaluating 2-DNF Formulas on Ciphertexts'. In: *International Conference on Theory of Cryptography*. 2. IACR. 2005, pp. 325–341. DOI: 10.1007/978-3-540-30576-7_18.

[BGV12] Zvika Brakerski, Craig Gentry & Vinod Vaikuntanathan. '(Leveled) Fully Homomorphic Encryption without Bootstrapping'. In: *Innovations in Theoretical Computer Science Conference*. 3. ACM. 2012, pp. 309–325. DOI: 10.1145/2090236.2090262.

[BLLN13] Joppe W. Bos, Kristin Lauter, Jake Loftus & Michael Naehrig. 'Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme'. In: *International Conference on Cryptography and Coding*. 14. IMA. 2013, pp. 45–64. DOI: 10.1007/978-3-642-45239-0_4.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim & Yongsoo Song. 'Homomorphic Encryption for Arithmetic of Approximate Numbers'. In: *International Conference on the Theory and Application of Cryptology and Information Security*. 23. IACR. 2017, pp. 409–437. DOI: 10.1007/978-3-319-70694-8_15.

[Cle+15] Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren & Ingrid Verbauwhede. 'Efficient Software Implementation of Ring-LWE Encryption'. In: *Design, Automation & Test in Europe Conference & Exhibition*. IEEE. 2015, pp. 339–344. DOI: 10.7873/DATE.2015.0378.

[Dij+10] Marten van Dijk, Craig Gentry, Shai Halevi & Vinod Vaikuntanathan. 'Fully Homomorphic Encryption over the Integers'. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. 29. IACR. 2010, pp. 24–43. DOI: 10.1007/978-3-642-13190-5_2.

[DM98] Leonardo Dagum & Ramesh Menon. 'OpenMP: An Industry-Standard API for Shared-Memory Programming'. In: *IEEE Computational Science and Engineering* 5.1. 1998, pp. 46–55. DOI: 10.1109/99.660313.

[ElG85] Taher ElGamal. 'A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms'. In: *IEEE Transactions on Information Theory* 31.4. IEEE, 1985, pp. 469–472. DOI: 10.1109/TIT.1985.1057074.

[Gen09] Craig Gentry. 'Fully Homomorphic Encryption Using Ideal Lattices'. In: *Symposium on the Theory of Computing*. 41. ACM. 2009, pp. 169–178. DOI: 10.1145/1536414.1536440.

[GHS15] Craig Gentry, Shai Halevi & Nigel P. Smart. *Homomorphic Evaluation of the AES Circuit (Updated Implementation)*. 2015. IACR: 2012/099/20150103:190644.

[GSW13] Craig Gentry, Amit Sahai & Brent Waters. 'Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based'. In: *International Conference on Advances in Cryptology*. 33. IACR. 2013, pp. 75–92. DOI: 10.1007/978-3-642-40041-4_5.

[Har97] David A Harville. 'Kronecker Products and the Vec and Vech Operators'. In: *Matrix Algebra From a Statistician's Perspective*. §16. Springer, 1997, pp. 337–378. DOI: 10.1007/0-387-22677-X_16.

[HEAAN] Andrey Kim, Kyoohyung Han, Miran Kim & Yongsoo Song. *HEAAN*. Cryptography LAB in Seoul National University. URL: https://github.com/snucrypto/HEAAN.

[HElib] Shai Halevi & Victor Shoup. *HElib*. URL: https://github.com/homenc/HElib.

[HEXL] Fabian Boemer, Sejun Kim, Gelila Seifu et al. *HEXL*. Intel Corporation. URL: https://github.com/intel/hexl.

[HPS98] Jeffrey Hoffstein, Jill Pipher & Joseph H Silverman. 'NTRU: A Ring-Based Public Key Cryptosystem'. In: *International Symposium on Algorithmic Number Theory*. 3. 1998, pp. 267–288. DOI: 10.1007/BFb0054868.

[Huy20] Daniel Huynh. *CKKS Explained: Part 1, Full Encoding and Decoding*. OpenMined. 1st Sept. 2020. URL: https://blog.openmined.org/ckks-explained-part-1-simple-encoding-and-decoding/.

[Jia+18] Xiaoqian Jiang, Miran Kim, Kristin Lauter & Yongsoo Song. 'Secure Outsourced Matrix Computation and Application to Neural Networks'. In: *Conference on Computer and Communications Security*. 25. ACM. 2018, pp. 1209–1222. DOI: 10.1145/3243734.3243837.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan & Anantha Chandrakasan. 'GAZELLE: A Low Latency Framework for Secure Neural Network Inference'. In: *USENIX Security Symposium*. USENIX. 2018, pp. 1651–1669. ISBN: 978-1-939133-04-5.

[Knu92] Donald E Knuth. 'Two Notes on Notation'. In: *The American Mathematical Monthly* 99.5. Taylor & Francis, 1992, pp. 403–422. ARXIV: math/9205211.

[Lattigo] Christian V. Mouchet, Jean-Philippe Bossuat, Juan R. Troncoso et al. *Lattigo*. EPFL-LDS. URL: https://github.com/ldsec/lattigo.

[LM21] Baiyu Li & Daniele Micciancio. 'On the Security of Homomorphic Encryption on Approximate Numbers'. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. 40. IACR. 2021, pp. 648–677. DOI: 10.1007/978-3-030-77870-5_23.

[LPR13] Vadim Lyubashevsky, Chris Peikert & Oded Regev. 'On Ideal Lattices and Learning with Errors over Rings'. In: *Journal of the ACM* 60.6. ACM, 2013, 35 pp. DOI: 10.1145/2535925.

[LTV12] Adriana López-Alt, Eran Tromer & Vinod Vaikuntanathan. ‘On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption’. In: *Symposium on the Theory of Computing*. 44. ACM. 2012, pp. 1219–1234. DOI: 10.1145/2213977.2214086.

[McI16] James McIvor. ‘Ring Theory’. In: *Math 113*. Aug. 2016.

[Mil07] Elizabeth Million. ‘The Hadamard Product’. 12th Apr. 2007.

[Pai99] Pascal Paillier. ‘Public-Key Cryptosystems Based on Composite Degree Residuosity Classes’. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. 17. IACR. 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16.

[PALISADE] Gerard Ryan, Dave Cousins, Kurt Rohloff & Yuriy Polyakov. *PALISADE*. URL: https://gitlab.com/palisade/palisade-release.

[RAD78] Ronald L. Rivest, Len Adleman & Michael L. Dertouzos. ‘On Data Banks and Privacy Homomorphisms’. In: *Foundations of Secure Computation*. Academic Press, 1978, pp. 169–180.

[Reg05] Oded Regev. ‘On Lattices, Learning with Errors, Random Linear Codes, and Cryptography’. In: *Symposium on the Theory of Computing*. 37. ACM, 2005, pp. 84–93. DOI: 10.1145/1060590.1060603.

[RSA78] Ronald L. Rivest, Adi Shamir & Leonard Adleman. ‘A Method for Obtaining Digital Signatures and Public-Key Cryptosystems’. In: *Communications of the ACM* 21.2. ACM, 1978, pp. 120–126. DOI: 10.1145/359340.359342.

[SEAL] Kim Laine, Wei Dai, Radames Cruz Moreno et al. *Microsoft SEAL*. Release 3.7.2. Microsoft Research. Nov. 2021. URL: https://github.com/Microsoft/SEAL.

[Yao82] Andrew C. Yao. ‘Protocols for Secure Computations’. In: *Symposium on Foundations of Computer Science*. 23. IEEE. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.88.

[zlib] Jean-loup Gailly & Mark Adler. *zlib*. URL: https://github.com/madler/zlib.

[ZPG15] Efstathios Zachos, Aristeidis Pagourtzis & Panagiotis Grontas. *COMPUTATIONAL CRYPTOGRAPHY*. Greek. 12 vols. Kallipos, 2015. ISBN: 978-960-603-276-9. HDL: 11419/5439.

[zstd] Yann Collet. *Zstandard*. Facebook. URL: https://github.com/facebook/zstd.