HOMOMORPHIC ENCRYPTION APPLICATIONS USING PALISADE
Best Practices for Building HE Solutions with Application to Privacy-Preserving GWAS

Yuriy Polyakov
ypolyakov@dualitytech.com
PREREQUISITES FOR THIS TALK

• Sasha’s talk (right before this one)
• Webinar #6: Introduction to Approximate Homomorphic Encryption
  (https://www.youtube.com/watch?v=_s1B128sql)
• Other related recorded webinars can be accessed at https://palisade-crypto.org/webinars/
AGENDA

• Data science guidelines
• Data encoding optimizations
• Operation re-ordering & other “compiler” optimizations
• System-level optimizations
DATA SCIENCE GUIDELINES

• Guidelines for developing efficient data science algorithms for the problem
  • What algorithm gives us the solution using the least multiplicative depth?
    • Parallelizable workflows are always preferred over sequential ones to use a smaller multiplicative depth
  • What algorithm minimizes or avoids HE-hard operations, such as comparisons?
  • How can we utilize SIMD packing to increase the throughput/achieve parallelization?
  • Related to SIMD, how to minimize the number of rotations as they are expensive?
  • What level of information leakage is allowed?

• Most of the work is done in the clear, before the first HE prototype is written
  • Always verify the algorithms and check the quality of results using standard data science packages and metrics
  • The output precision is often not the right metric

• For the GWAS problem, the Chi-square test was the best option
  • It requires only a multiplicative depth of 3
  • All operations are SIMD-friendly and do not require any rotations
  • No comparisons are needed
DATA ENCODING OPTIMIZATIONS

- How to best encode the application data structures into HE plaintexts?
  - How to encode matrices and vectors to support efficient matrix arithmetic?
  - **Packed-matrix encoding**

\[
X = \begin{bmatrix}
X_{11} & X_{12} & \ldots & X_{1k} \\
X_{21} & X_{22} & \ldots & X_{2k} \\
\vdots & \vdots & \ddots & \vdots \\
X_{N1} & X_{N2} & \ldots & X_{Nk}
\end{bmatrix}
\]

\[
X^\top = \begin{bmatrix}
X_{11} & X_{11} & \ldots & X_{11} \\
X_{12} & X_{12} & \ldots & X_{12} \\
\vdots & \vdots & \ddots & \vdots \\
X_{1k} & X_{1k} & \ldots & X_{1k} \\
\vdots & \vdots & \ddots & \vdots \\
X_{N1} & X_{N1} & \ldots & X_{N1} \\
X_{N2} & X_{N2} & \ldots & X_{N2} \\
\vdots & \vdots & \ddots & \vdots \\
X_{Nk} & X_{Nk} & \ldots & X_{Nk}
\end{bmatrix}
\]

\[
y = \begin{bmatrix}
y_1 & y_1 & \ldots & y_1 \\
y_2 & y_2 & \ldots & y_2 \\
\vdots & \vdots & \ddots & \vdots \\
y_N & y_N & \ldots & y_N
\end{bmatrix}
\]

- **Packed-integer encoding**: same values is cloned to all slots
- The use of these two methods of encoding significantly reduces the cost of matrix arithmetic (number of key switching operations)
OPERATION RE-ORDERING & OTHER “COMPILER” OPTIMIZATIONS

- Most expensive operations in CKKS:
  - Key switching operations (we need to perform key switching after multiplications and rotations)
    - Key switching after multiplication is called relinearization
  - Rescaling operations

- We can reduce the number of both operations by utilizing the so-called lazy relinearization and lazy rescaling
  - The idea is that we do not need to apply relinearization or rescaling immediately but can apply it after, for example, the results of several multiplications are added together

- We can take advantage of fast rotations, which do not need the full key switching operation, when we need multiple rotations of the same ciphertext
  - We can precompute the common expensive part (once) and then reuse it for multiple rotations

- Take advantage of binary tree multiplication to reduce the depth, i.e., execute a chained product in a certain order

- Use closed-form expressions, i.e., unroll the loops

- Rewrite rotations used for summation over a vector as additions
SYSTEM-LEVEL OPTIMIZATIONS

• Utilize multithreading for loop parallelization (OMP is used in PALISADE)
  • Loop parallelization is typically most effective at higher layers (closer to the application)
• Serialize/deserialize large structures with keys or ciphertexts at a more granular level to keep RAM utilization relatively low
• Encrypt the ciphertexts at the first level used
• Compress the evaluation keys as needed (including during the computation)
• Reduce the number of rotation keys if RAM is limited
• Distribute the computation over multiple computer systems keeping communication costs relatively small
MORE INFORMATION

• Source code: https://gitlab.com/duality-technologies-public/palisade-gwas-demos/
• PNAS Paper: https://www.pnas.org/content/117/21/11608
THANK YOU

ypolyakov@dualitytech.com

https://palisade-crypto.org