Private Predictions with Homomorphic Encryption

Kim Laine
kim.laine@microsoft.com
Presentation Outline

Privacy in Prediction

SEAL

Examples
Privacy in Prediction
Wait! What about Privacy?
Who else is going to see your DNA sequence and the prediction?
Who else is going to see your DNA sequence and the prediction?

“Sorry, your DNA does not match this job description.”

“Here is an advertisement that according to your DNA you will not be able to resist.”

“We are not giving you this loan because it is not in your DNA to pay it back.”
Can encryption help?
Can encryption help?

Possibly. But need *very* special type of encryption!
Inference over encrypted data
Inference over encrypted data
Inference over encrypted data
Can encryption help?

Possibly. But need very special type of encryption!
Can encryption help?

Possibly. But need very special type of encryption!

Yes. Homomorphic encryption.
Fully Homomorphic Encryption Using Ideal Lattices

Craig Gentry
Stanford University and IBM Watson
cgentry@cs.stanford.edu

ABSTRACT

We propose a fully homomorphic encryption scheme – i.e., a scheme that allows one to evaluate circuits over encrypted data without being able to decrypt. Our solution comes in three steps. First, we provide a general result – that to construct an encryption scheme that permits evaluation of arbitrary circuits, it suffices to construct an encryption scheme that can evaluate (slightly augmented versions of) its own decryption circuit; we call a scheme that can evaluate its (augmented) decryption circuit bootstrappable.

Next, we describe a public key encryption scheme using ideal lattices that is almost bootstrappable. Lattice-based cryptosystems typically have decryption algorithms with low circuit complexity of standard circuits (e.g., integer addition), but are produced by Rivest, Adleman and Dertouzos [54] shortly after the invention of RSA by Rivest, Adleman and Shamir [55]. Basic RSA is a multiplicatively homomorphic encryption scheme – i.e., given RSA public key $pk = (N, e)$ and ciphertexts $\{\psi_i \leftarrow \pi_i^e \mod N\}$, one can efficiently compute $\Pi_i \psi_i = (\Pi_i \pi_i)^e \mod N$, a ciphertext that encrypts the product of the original plaintexts. Rivest et al. [54] asked a natural question: What can one do with an encryption scheme that is fully homomorphic: a scheme $\mathcal{E}$ with an efficient algorithm Evaluate$_{\mathcal{E}}$ that, for any valid public key $pk$, any circuit $C$ (not just a circuit consisting of multiplications and gates), and any ciphertexts $\psi_i \leftarrow$ Encrypt$_{\mathcal{E}}(pk, \pi_i)$, output $\psi \leftarrow$ Evaluate$_{\mathcal{E}}(pk, C, \psi_1, \ldots, \psi_t)$.
Private data $X$ → Encrypt → $Enc(X)$ → Compute function $F$ → $F(Enc(X))$ → Decrypt → $F(X)$
Compute function $F$

Encrypt

Private data $X$

$\text{Enc}(X)$

$F(\text{Enc}(X))$

Data owner can read result

Service provider sees only encrypted data
Private data $X$

Encrypt

$Enc(X)$

Compute function $F$

$F(Enc(X))$

F(X) must be a polynomial in the data $X$

Service provider sees only encrypted data

Data owner can read result
Simple Encrypted Arithmetic Library – SEAL

Easy-to-use homomorphic encryption library
Homomorphic encryption library by MSR Cryptography Research group
Focus on ease-of-use, good API design, good engineering
Written in C++11
Contains .NET wrappers for entire public API
Source code publicly available
Under active development

http://sealcrypto.codeplex.com
void simple_example()
{
    EncryptionParameters parms;
    parms.poly_modulus() = "1x^2048 + 1";
    parms.coeff_modulus() = ChooserEvaluator::default_parameter_options().at(2048);
    parms.plain_modulus() = 1 << 10;

    KeyGenerator keygen(parms);
    keygen.generate();
    auto public_key = keygen.public_key();
    auto secret_key = keygen.secret_key();

    BinaryEncoder encoder(parms.plain_modulus());
    Encryptor encryptor(parms, public_key);
    auto plain1 = encoder.encode(5);
    auto plain2 = encoder.encode(7);
    auto enc1 = encryptor.encrypt(plain1);
    auto enc2 = encryptor.encrypt(plain2);

    Evaluator evaluator(parms);
    auto enc_product = evaluator.multiply(enc1, enc2);
    auto enc_sum = evaluator.add(enc1, enc2);

    Decryptor decryptor(parms, secret_key);
    auto plain_product = decryptor.decrypt(enc_product);
    auto plain_sum = decryptor.decrypt(enc_sum);
    uint64_t product = encoder.decode_uint64(plain_product);
    uint64_t sum = encoder.decode_uint64(plain_sum);

    cout << product << " " << sum << endl;
}
Examples
Genomic Predictor

Sending Request

Server Processing

Decrypting Results

Clear

Next

37.2

200K SNPs
Demo: Pneumonia Risk Prediction
Thank you!

Contact: kim.laine@microsoft.com