Charm: A Framework for Rapidly Prototyping Cryptosystems

Joseph A. Akinyele, Matthew D. Green, and Aviel D. Rubin
Johns Hopkins University
{akinye1j, mgreen, rubin}@cs.jhu.edu

1 Abstract

We present Charm, an extensible Python-based framework for rapidly prototyping cryptographic systems. Charm was designed from the ground up to support the development of advanced cryptographic schemes and protocols. It includes support for multiple cryptographic settings such as integer and elliptic curve groups. Moreover, we provide an extensive library of re-usable routines such as secret sharing and hash functions, and the infrastructure necessary to quickly implement interactive protocols.

Our framework, shown in Figure 1, utilizes a developer-friendly high-level language with operator overloading so that algorithms can be expressed as mathematical expressions. In addition, we take advantage of automatic memory management and typing features to reduce programmer errors. Our goal is to provide a tool that allows developers and researchers to quickly implement advanced cryptographic algorithms while alleviating them of low-level implementation details such as with C or C++.

Charm provides a series of message space conversion routines that enable different cryptosystems to interoperate. In addition, we provide support for embedding cryptosystems in existing applications. This includes support for automatically handling serialization of public/private keys and ciphertexts. We have implemented several cryptographic algorithms in the research literature. We also implemented the first (to the best of our knowledge) implementations of new cryptographic schemes such as multi-authority ciphertext-policy attribute-based encryption, a stateful hash-and-sign RSA digital signature, and anonymous ring/group signatures. Our techniques result in an order of magnitude reduction in code size with only a negligible impact to performance.

The Charm framework is modular with low-level native modules implementing performance critical details and high-level modules providing advanced features. Computationally intensive operations are encapsulated in C math libraries and exposed in Charm via the Python/C extensions API. We provide a benchmarking module for schemes and protocols to measure efficiency. Furthermore, we provide a protocol engine that handles arbitrary data serialization and network transmission between interactive parties. This allows developers to focus on the details of their protocol rather than on low-level network implementation specifics.

Charm is in its 3rd alpha release, and the first beta release is planned for mid-February 2012. The code and a full paper describing the system are available at http://charm-crypto.com.

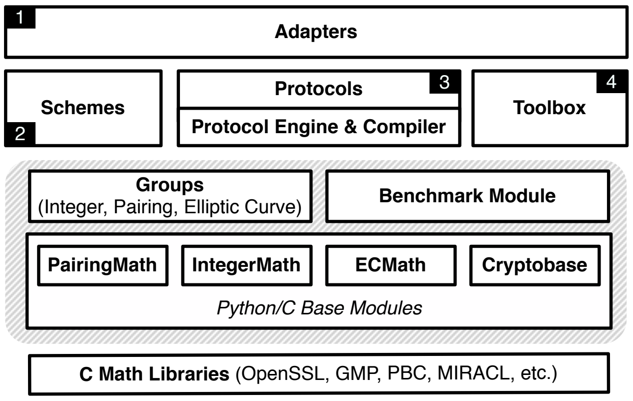

Figure 1: Charm architecture overview. There are four major components built on top of Charm: adapters, schemes, a protocol engine, and a toolbox. 1) An adapter interface that alters the input/output or security properties of a scheme. This promotes code re-use by removing incompatibilities between implementations. 2) A library of implemented cryptosystems accessed via standard APIs. 3) Protocols infrastructure to support the development of interactive protocols via a dedicated protocol engine. 4) An extensible library of common routines, including secret sharing, X.509 certificate handling, parameter generation, policy parsing and hash functions.

This work was sponsored by NSF Award CNS1010928 and HHS Grant Number 90TR0003/01.