RSA and ECC in JavaScript

The jsbn library is a fast, portable implementation of large-number math in pure JavaScript, enabling public-key crypto and other applications on desktop and mobile browsers.

Demos

- RSA Encryption Demo - simple RSA encryption of a string with a public key
- RSA Cryptography Demo - more complete demo of RSA encryption, decryption, and key generation
- ECDH Key Agreement Demo - Diffie-Hellman key agreement using elliptic curves

Source Code

The API for the jsbn library closely resembles that of the `java.math.BigInteger` class in Java. For example:

```
x = new BigInteger("abcd1234", 16);
y = new BigInteger("beef", 16);
z = x.mod(y);
alert(z.toString(16));
```

will print `b60c`.

Core Library

- jsbn.js - basic BigInteger implementation, just enough for RSA encryption and not much more.
- jsbn2.js - the rest of the library, including most public `BigInteger` methods.

RSA

- rsa.js - implementation of RSA encryption, does not require `jsbn2.js`.
- rsa2.js - rest of RSA algorithm, including decryption and keygen.

ECC

- ec.js - elliptic curve math, depends on both `jsbn.js` and `jsbn2.js`
- sec.js - standard elliptic curve parameters

Utilities

- rng.js - rudimentary entropy collector and RNG interface, requires a PRNG backend to define `prng_newstate()`.
- prng4.js - ARC4-based PRNG backend for `rng.js`, very small.
- base64.js - Base64 encoding and decoding routines.
- sha1.js - SHA-1 hash function, only needed for IBE demo.

Interoperability

The demo encrypts strings directly using PKCS#1 encryption-style padding (type 2), which is currently the only supported format. To show interoperability with a potential OpenSSL-based backend that decrypts strings, try the following on any system with the OpenSSL command line tool installed:

1. Generate a new public/private keypair:

   ```
   $ openssl genrsa -out key.pem
   Generating RSA private key, 512 bit long modulus
   ..++++++++++++
   ..............++++++++++++
   e is 65537 (0x10001)
   $
   ```

2. Extract the modulus from your key:

   ```
   $ openssl rsa -in key.pem -noout -modulus
   Modulus=DA3BB4C40E3C7E76F7DBDD8BF3DF0714CA39D3A0F7F9D7C2E4FEDF8C7B28C2875F7EB98950B22AE82D539C1ABC1AB550BA0B2D52E3EF7BDFB78A5E817D74BBDB
   $
   ```

3. Go to the RSA Encryption demo and paste the modulus value into the "Modulus (hex)" field at the bottom.
4. Make sure the value in the "Public exponent" field is "10001", or whatever value your public key uses.
5. Type in a short string (e.g. `testing`) into the "Plaintext (string)" field and click on "encrypt". The result should appear in the "Ciphertext" fields.
6. Copy the base64 version of the ciphertext and paste it as the input of the following command:

   ```
   $ openssl base64 -d | openssl rsautl -inkey key.pem -decrypt
   1JW24UMKntVhmmDilAYC1AjLxgiWHBzTzZsCVAejLjVri92abLHkSyLisVyAdYVr
   fiS7FchtI9vupe9JF/m3Kg==
   ```

   Hit ctrl-D or whatever your OS uses for end-of-file. Your original plaintext should appear:

   ```
   testing$
   ```

Performance

The speed tables contain detailed timing information for `jsbn` performing public-key operations such as RSA, ECC, and IBE.

Projects that use jsbn

- Forge - a pure JavaScript implementation of SSL/TLS, includes a discussion of their choice of BigInteger library
- Dojo Toolkit uses jsbn in their dojox.math.BigInteger class.
- No More Cleartext Passwords - this project switched from another JavaScript BigInteger library for performance reasons
- Google's V8 benchmark suite, version 6
- JavaScript Cryptography Toolkit
- RSA-Sign JavaScript library
- JavaScript RSA

History

Version 1.2 (3/29/2011):

Added `square` method to improve ECC performance.
Use randomized bases in `isProbablePrime`

Version 1.1 (9/15/2009):

Added support for utf-8 encoding of non-ASCII characters when PKCS1 encoding and decoding JavaScript strings.
Fixed bug when creating a new BigInteger("0") in a non power-of-2 radix.

Licensing

`jsbn` is released under a BSD license. See `LICENSE` for details.

Tom Wu
Last modified: Tue Sep 15 23:30:00 PST 2009