FIPS Certification and the Bouncy Castle Project, or What Do You Mean I Can’t Just Do a New Release Tomorrow?
A Quick Look at Bouncy Castle

• Collection of APIs in Java and C#

• Provides basic cryptography and implementations for a range of ISO, ITU-T, and IETF standards such as X.509, OpenPGP, S/MIME, and TLS.

• Core Java API 324,000 lines (565,000 including tests and compatibility APIs).

• Core C# API 165,000 lines (237,000 including tests)
History of Bouncy Castle

- Started in 2000
- Originally just Java, C# added in 2004
- 3 core developers and a bit of extra help

- Core feature set determined by essential and interesting algorithms at the time
- Algorithm set evolved according to the needs of the APIs built on the core crypto library
- APIs evolved according to core developer needs and user feedback
History of Bouncy Castle (Cont.)

- Original Java API was 27,000 lines
- By around 2007 the effect of adding around 300,000 lines of code in an ad-hoc fashion was starting to be felt
- Started grappling with the issue of whether to simply run from the project, or find some way of funding it
- Decided not to run, but needed some way of moving the dialogue with our users to the fact the work needed to be paid for
Interest in FIPS 140-2

- We started getting requests for FIPS compliance as soon as the APIs became commonly accepted.
- FIPS 140-2 also provides the basis for many other standards.
- And, put bluntly, everyone understands FIPS 140-2 has to be paid for, so it provided a way of starting to move to the funding dialogue.
- Provided a focus point for raising the quality of the APIs as well.
Launching the FIPS effort

- Had enough money to do an initial product and documentation review, so started there

- Other expenses: lab fees, NIST recovery fees, and the actual cost of doing it

- Preparation for product review showed there were a number of problems with the way BC did things, mainly in the low level, or “light weight” APIs

- Low-level BC was designed for maximum flexibility

- FIPS 140-2 is, in some ways, more protective
Hindsight

• As it turned out, raising the money, while it took time, was not a huge problem
• Even revamping the low-level API was not a massive effort
• Issues around algorithm testing are easy to deal with as well
• Operational testing, while still more “hands on” than we would like, was not a problem item either
• The update process, on the other hand was, and still is, a killer
Updates – Non-FIPS

- BC developer either gets inspired, receives a code contribution, or flies into a panic about something
- Makes changes, writes tests, does a full build to do regression testing
- Automated build (Jenkins) takes over and runs some more builds doing further testing
- Beta release gets shipped for Java 1.5 and later
- Eventually an actual release gets done
Updates – Non-FIPS (Cont.)

- Record for receipt of report to working tested beta with fix is around 2 hours
- Record for roll out of a full release across all the JVMs supported is around 16 hours
- Further work on automation could reduce the roll out time of a full release
- Overall user satisfaction quite high
Updates - FIPS

- Two categories – before submission, after submission
- Submission represents CAVP and operational testing
- Record for before submission turnaround is about the same as for Non-FIPS
- Once CAVP testing and operational testing are done it suddenly gets a lot harder. Worst case is that a modification results in a full resubmission.
Culture Issues – Development

- Independent Open Source projects are resource constrained.
- A project's survival depends on successfully doing “as needed” development.
- “as needed” means that you do not do things fully up front, opting instead for an evolutionary approach so features are added as required.
- “as needed” is also one of the principles of agile development.
Culture Issues - Testing

- “as needed” also affects testing
- Observer bias – difficult to look in depth at things that you either do not think of or do not care about
- Observer bias is an issue for the library developers
- Correct behaviour which has arisen implicitly from otherwise tested code often not captured in explicit tests
- Identifying bias requires testing tests
Culture Issues – Testing (Cont.)

• Likewise, user acceptance testing also suffers from observation bias.
• Users generally only care about what works for them.
• Reporting of issues varies according to state of project
• The reality is everyone feels they are short of time
• End result is things get missed because they are “not important”
• Traditionally testing not well budgeted for even in closed source efforts
FIPS Process

- FIPS certification is managed as a “big up front” process
- From a FIPS perspective CAVP and operational testing guarantee algorithms are correct and health tests are in place
- From a user perspective CAVP and operational testing do not show that the scaffolding that goes around the algorithms is correct
- Unfortunately both the algorithms and the scaffolding end up inside the implementation boundary of a software module
Why this Matters

• For API users, algorithm correctness is also important, but so is the correctness of the scaffolding.

• In the case of the JCA/JCE scaffolding correctness is in two forms: compliance to that which is documented, and compliance to that which is implemented already (so implied).

• In some cases 3rd party vendors assume things about scaffolding which can make an API unusable in an application if the API does not conform.
Changes in Approach - BC

• Needing to write and maintain more tests to cover broad scaffolding issues and test for conformance

• Had to get over misplaced faith in code coverage

• Code coverage becomes less useful as a guide to code health: both branches and combination of branches need to be captured, and also return types (e.g. keys implementing particular interfaces)
Changes in Approach – BC (Cont.)

- Review work needed to take “big up front” into consideration
- Originally questions were largely around a particular algorithm and broader integration followed
- Now questions need to be raised early about broader integration as well
- Tried to explicitly allow time and budget for broader testing
End Result

• Luckily we had a bit of prior experience, however the user community of the general APIs and even to a degree the FIPS user community followed the “it works for me” principal.

• Biggest problem was testing unwritten “standards” in the JCE/JCA properly.

• 28 issues were identified with 1.0.0 in the first 12 months – 2 of these touched on code in the CAVP tested implementation classes.

• 12 issues could not be worked around – only 1 of these touched on code in the CAVP tested implementation classes.
Things to Think About

- First, automated testing would help. See:
  https://github.com/usnistgov/ACVP
  https://github.com/usnistgov/AMVP

- As far as software modules go, the boundary idea appears to be based around assembler and a single loadable library

- For higher level languages like C# and Java it should be possible to safely reduce the boundary to avoid a full change process further
Things to Think About (Cont.)

- The actual safety with which a change can be made depends on a number of things in, and about, a software module
- Some consideration of the nature of the build process used to develop the software module should be made as well
Who Makes the Call?

Labs are already responsible for code review of non-CAVP tested code.

It might make sense for them to be able to make a call on boundary with modern languages and development processes.
Another Reason to Worry
So Finally...

- We are starting to enter a period where change is likely to become the new normal
- It may not be right to assume we can go through a 12-24 month process to get something out the door!
Thanks for listening.

Questions?