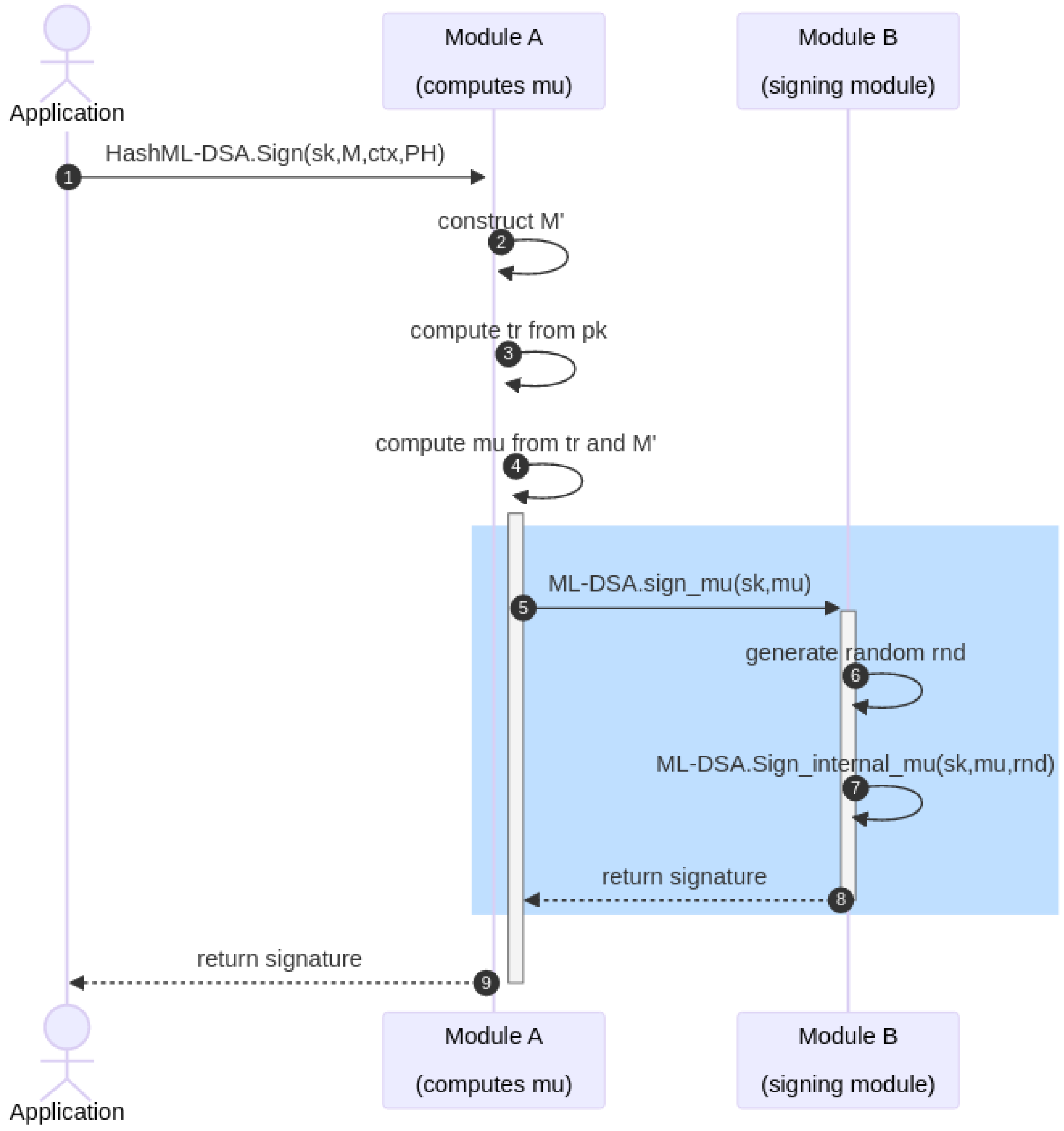
Q:

FIPS 204 Section 6 states that the internal interfaces in this section *should not* be made available to applications, and that the interfaces in Section 5 *should* be used instead. The interfaces in Section 5 are not sufficient to allow an implementation to compute mu externally and provide it to the signing implementation. Will the Cryptographic Module Validation Program allow this in FIPS validated modules, and how?

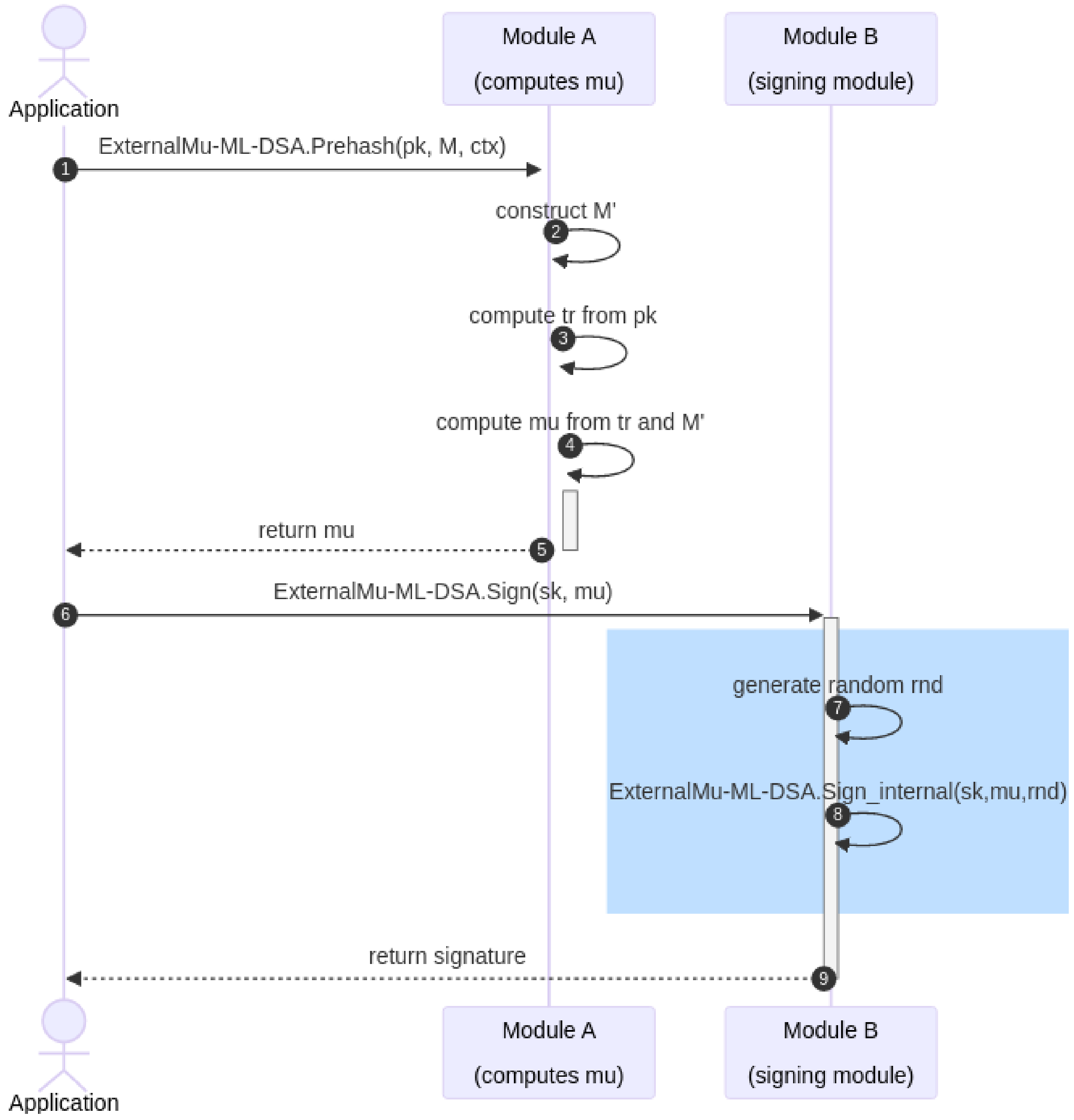
A:

Computing mu in a different cryptographic module is explicitly permitted by FIPS 204 (see note on step 6 of Algorithm 7 or note on step 7 of Algorithm 8). In cases where something explicitly permitted by FIPS 204 requires a different interface from the ones specified, this interface may be provided. Implementations may use a different interface than defined in Algorithm 7 (or Algorithm 8) FIPS 204 when signing (or verifying) a mu computed externally. Example interfaces are provided below. The appropriate Cryptographic Algorithm Validation Program testing must be run on the signing (or verifying) module and on the module that computes mu. The signing (or verifying) module must complete the ML-DSA SigGen (or ML-DSA SigVer) FIPS 204 tests with the modified internal signature interface along with any prerequisite algorithm tests. The module that computes mu must, at minimum, have a validated implementation of SHAKE256 along with any hash functions used as part of a pre-hash external interface.

For example, as shown in the diagram below, module A may implement ML-DSA.Sign(sk, M, ctx) or HashML-DSA.Sign(sk, M, ctx, PH). Module A's implementation of either function may call ML-DSA.Sign_mu(sk, mu), implemented on module B, which may then generate a random value rnd and call ML-DSA.Sign_internal_mu(sk, mu, rnd). To perform a signature, module A would construct M', compute tr from pk, compute mu from tr and M', then call module B via ML-DSA.Sign_mu(sk, mu) to obtain the signature. Module A would be the module that computes mu, and module B would be the signing module. In this case sk may not be the private key but a reference to it if it is already available on module B. If module B only contains a single private key, sk may be omitted from the interface.



Another example as shown in the diagram below, is a slight modification to the above example. Instead of module A receiving sk (or its reference) and directly interfacing with Module B, Module A may provide mu to a (possibly non-module) application, and that application would provide mu and sk (or its reference) to Module B that will intern generate the signature using an internal approved random number generator and return that signature to the application.


Application
Module A
(computes mu)
Module B
(signing module)
1 ExternalMu-ML-DSA.Prehash(pk, M, ctx)
2 construct M'
3 compute tr from pk
4 compute mu from tr and M'
5 return mu
6 ExternalMu-ML-DSA.Sign(sk, mu)
7 generate random rnd
8 ExternalMu-ML-DSA.Sign_internal(sk,mu,rnd)
9 return signature

An even further modification to the above example could be if Module A only implemented SHAKE256 and the hashes and XOFs needed to pre-hash M. In this case, an application outside of either module may use calls to Module A to pre-hash M as part of constructing M', to compute tr [from pk] and to compute mu [from BytesToBits(tr)|| M']. Module B would still implement the approved random number generator and implement the same CAVP-tested modified internal interface shown above.